Title
Predicting Fuel Salt Composition via Linear Optimization in Molten Salt Reactors

Author
Wooten, Daniel D

Publication Date
2019

Peer reviewed|Thesis/dissertation

Predicting Fuel Salt Composition via Linear Optimization in Molten Salt Reactors

by

Daniel David Wooten

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Nuclear Engineering

and the Designated Emphasis

in

Computational and Data Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Massamiliano Fratoni, Chair
Assistant Professor Rachel Slaybaugh
Professor James Demmel

Fall 2019

Predicting Fuel Salt Composition via Linear Optimization in Molten Salt Reactors

Abstract

Predicting Fuel Salt Composition via Linear Optimization in Molten Salt Reactors

by

Daniel David Wooten

Doctor of Philosophy in Engineering – Nuclear Engineering
and the Designated Emphasis
in
Computational and Data Science and Engineering

University of California, Berkeley

Associate Professor Massamiliano Fratoni, Chair

Molten salt reactors (MSRs) are a class of nuclear reactor which uses a molten ionic liquid as either the coolant or also as the fuel. While a 8 MWth MSR was successfully operated in the 1960s it was not until the early 2000s that MSRs gained widespread attention. Since then MSRs have enjoyed plentiful research support. Despite such support only one general MSR fuel cycle analysis tool is available for use by the research community and even this this tool lacks features necessary for modelling a MSR, and so possibly providing answers of a lower quality.

In this work a method is proposed and implemented within the SERPENT 2 reactor physics Monte-Carlo code. This method, named ADER - the Advanced Depletion Extension for Reprocessing - is a seamlessly incorporated source code modification to the SERPENT 2 base code which allows the user to define arbitrary collections of elements, isotopes, and chemicals as well as relationships among them.

Furthermore ADER allows the user to specify a variety of mass flows subject to the constraints as defined through the collections of elements, isotopes, and chemicals the user has structured. Along with support for constraints involving both corrosion and nuclear control concerns, ADER allows the user to optimize the solution against a quantity of interest - e.g, total uranium fed into the system.

Through these structures much of the complex chemistry, corrosion modelling, and nuclear concerns of operating a MSR can be linearized and solved against an optimization target, which is necessary given the large number of constraints and variables in such a problem space. Linearization reduces the problem complexity and eliminates concerns over local versus global optimization targets. Within the typically narrow operating parameters of MSRs linearization is an appropriate approximation to the higher dimensional equations representing the phenomenon involved.

This linear system and optimization target may then be passed to a linear optimization solver, in this work the CLP library as part of the COIN-OR package, from which an optimized system material composition and material flows solution may be found. ADER then uses this solution to create a brand new depletion matrix which SERPENT 2 then solves using the CRAM approximation method.

From this algorithm a more accurate modelling of MSR fuel cycles and physics may be arrived at through the consideration of chemistry driven limitations, corrosion driven limitations, nuclear driven limitations, and operator driven limitations. Results from this implemented method indicate that ADER drives the MSR fuel cycle simulations towards a more physically representative result. Unfortunately, as detailed later in this work, an underlying and pernicious numerical instability issue was uncovered within the linear optimization library selected for this work. Any future work on this method must begin with the adoption of a quadruple-precision floating-point linear optimization library over the current implementation of CLP as used in ADER.

In the following chapters an introduction to MSRs and their fuel cycle modelling is given. Following this the theory behind ADER and its implementation within SERPENT 2 is discussed after which the results from one of the less numerically unstable simulations is presented after which concluding remarks are given.

Contents

List of Figures

Acknowledgments

This work is dedicated to Garik Sadovy and Andrew Wieting - lights which blinked out too soon.

A Ph.D is awarded to one person but it takes a village to build one.

My rock and my foundation through this whole endeavor has been my family. No one could ask for better parents, my mother Lisa Wooten and my father David Wooten, nor for a better brother, Samuel Wooten. You all have celebrated my triumphs and supported me through the long dark nights of graduate school. Mom and Dad, I never felt fear this whole time. Trepidation, despair, and worry for sure - but not fear. I had no fear because I knew I couldn't fail you. My brother Sam, always you have filled me with hope. Always I could believe in the you who believed in me.

Kelly Rowland, you've always blazed the trail ahead of me, and have always been happy to turn around and show me the way. Your mentorship made me a better person and your friendship made Berkeley a home.

Dr. Katy Huff, you taught me that programming isn't just for wizards.

Michael Chapman, you've been and will be the older brother I never had and never knew I needed.

Christopher Lalau-Keraly, you taught me that life is for more than getting ahead.

May The Palace, our home and our love, live on in all of us.

Kristen Gallagher, you've shaped my life and do so to this day and my world is so full of color thanks to you.

Alice Kunin, you showed me the value I held in other people's eyes, thank you.

Sonya Magaña, you saved my hands and saved my PhD out of the kindness of your heart, a kindness you're never shy to share, and a kindness that makes the world a brighter place.

To the many friends and family who's tales are now entwined with this very journey, thank you for adding your thread to this tapestry.

To the puppy named Roo, thank you for making my thesis writing a stressless experience.

To the Daniel's family and Tivoli Cafe as well as the restaurant Tai San, thank you for feeding me along this journey.

To my advisor Max, thank you for giving me space to grow not just as an intellectual but as an individual.

This material is based upon work supported under an Integrated University Program Graduate Fellowship as well as supported by the Department of Energy under Award Number(s) DE-NE31310018M0025. This material is based upon work supported under a Nuclear Regulatory Commission Fellowship under Award Number(s) DE-NRC-HQ-84-15-G-0019. This material is based upon work supported under a Nuclear Science and Security Consortium Fellowship under Award Number(s) DE-NE3491-M2. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or limited, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at the University of California, Berkeley (supported by the UC Berkeley Chancellor, Vice Chancellor for Research, and Chief Information Officer).

Chapter 1

Introduction

Molten salt reactors (MSRs) first gained attention in the 1950s as a possible high power density propulsion source for a nuclear powered aircraft. While this idea never took flight, the possibility for molten salts as fuel for a nuclear reactor did in the form of the Molten Salt Reactor Experiment (MSRE) conducted at Oak Ridge National Laboratory (ORNL) during the 1960s and 70s [1]. Following an institutional redirection away from the MSRE program, molten salt studies continued at ORNL under the Molten Salt Breeder Reactor program through the 1970s and into the 80s [1]. After a brief lull in global interest during the late 80s and early 90s studies in MSRs began again in earnest in Europe beginning with the SAMOFAR program and continuing today under the ALISIA program [2]. Since this time global interest in MSRs has continued to grow with concepts coming from all corners of the globe from both industry and governments: the Molten Salt Actinide Recycler and Transmuter (MOSART) out of Russia, the FUJI series of reactors from Japan, the Liquid Fueled Thorium Molten Salt Reactor (LF-TMSR) out of China, the Molten Salt Fast Chloride Reactor (MCFR) from TerraPower in the United States, and many others. While global attention and efforts have also increased around a group of reactors which use molten salts without fissile content as a coolant, the fuel cycle analysis of these reactors is very well served by current simulation codes. In this work reference to molten salt reactors refers only to those reactors in which fission occurs dominantly in a liquid medium.

This global interest is due to molten salt's many promising characteristics such as high operating temperatures, low operating pressures (near atmospheric in most cases), excellent safety characteristics, and high fuel utilization just to name the top few. Many of these same characteristics create obstacles to modelling MSRs with today's computational tools due to the many physical differences between MSRs and today's more numerous solid fuel light water reactors. The key difference and originator of the challenges is that the fuel in a MSR is liquid and often flows through the core of the vessel and through a heat exchanger. This flow of fuel induces many effects such as the drift of delayed neutron precursors and the bubbling out of gaseous fission products, most notably the xenon isotopes.

Investigations of any nuclear reactor will include an analysis of the proposed fuel cycle. This is accomplished through coupling a transport code with a nuclear depletion code. Inves-

tigations of MSR fuel cycles are more challenging than those of their solid fuel counterparts due to a number of phenomena: the removal of various elements through natural processes such as bubbling and platting, and the incorporation of operator actions on the reactor fuel stream. Unlike light water reactors MSRs tend to operate near atmospheric pressure and their flowing fuel allows for the continual addition or removal of chemical species during reactor operation. This feature is exploited in two key ways.

First this feature allows MSRs to operate with a low excess reactivity as fissionable material may easily be added during operation. Secondly this feature allows the reactor operator to make adjustments to the liquid fuel salt composition. This is of critical importance as the liquid fuel salt in a MSR will have some desired chemical state in which the operator would like to maintain the salt. This is important for two reasons; first to prevent salt components from precipitating out of solution should they exceed their solubility limits, and second, the corrosion rate of the specialty structural nickel/iron alloys is on the order of micro-meters per year when the reduction potential of the salt is kept near a specific value while a slight deviation from this value can raise corrosion rates to centimeters per year [3]. As such, it is in the operator's best interest to keep a tight control over the MSR fuel salt.

Capturing all of these phenomena in a single nuclear fuel depletion code is non-trivial and raises the question, how does the fuel salt composition in a molten salt fueled reactor change over time in response to both nuclear fuel burnup and the reactor operator actions? Addressing this question is the goal of the work presented herein.

1.1 Framing the problem

The goal is to create a nuclear fuel depletion model which accounts for the physical phenomena acting on the liquid fuel of a MSR with a flowing fuel core. Perhaps the least quantified unknown is that of the operator's actions. What are the operators objectives? What tools does the operator have at their disposal? What are the measures by which the operator assess their actions? As in most simulation problems the variable of most variance here is the human. To approximate the human in this model they are replaced with a linear optimization routine. If the constraints facing the operator may be approximated with linear relationships and if the operator's desires may be approximated as minimizing or maximizing a given value then the choices of the operator may be predicted via linear optimization. Ensuring that the problem constraints and optimization targets are all linear strikes a compromise between the ease and reproducibility of the optimization solution and the model's adherence to the physical phenomena being simulated.

One goal an operator is stipulated to have is keeping specific chemical species within the fuel salt at some relative proportion to some other specific chemical species. This goal arises from the variable solubility of chemical species within a given salt mixture - a solubility which changes based both on temperature and the relative proportion of other chemical species. Despite the complex and often polynomial nature of many chemical solubility relationships, within a narrow window these relationships may be approximated as linear.

A related objective an operator is stipulated to have is to maintain the fuel salt reduction-oxidation (redox) potential at some desired value for corrosion prevention as mentioned above. The existence of a temperature gradient within the fuel flow loop of many MSR designs ensures that no equilibrium condition of dissolved ions will exist. As the fuel salt, for example, strips chromium from the structural alloys it would be possible for the reduced chromium ions to reach such a concentration that the chemical process of corrosion would come to a halt as chromium ions were stripped from the alloys as quickly as they plate back on. However, as a flowing fuel salt moves through a temperature gradient the solubility of various ions changes leading to the precipitation of these ions onto the cold - and occasionally hot - legs of the fuel loop. Having lost these precipitated ions the fuel salt is free to continue corroding structural alloys elsewhere in the reactor. As such the traditional means of corrosion prevention - sacrificial anodes, biasing the surface with an electric current, and controlling the activity of the corrosive species - are available to the MSR operator. Most commonly a reducing agent, such as beryllium metal, is added to the salt to balance the often oxidizing nature of fission. In this manner the salt redox potential is kept at a point which will inhibit corrosion - essentially there is very little free fluorine to pull out chromium and a slight excess of positive ions to catch any new free fluorine which fission may produce. Given that this redox potential is dependent on the power, spectrum, and salt in the reactor its maintenance requires constant attention which is accomplished via on-line redox potential measurement. Adjustment is accomplished with redox buffers, chemical additives which shift the chemical potential of the salt.

An important goal for all nuclear reactor operators is to maintain the multiplication value of the reacting system. In a traditional light water reactor this is accomplished by loading the reactor core with far more uranium than it needs to be critical and then controlling this excess criticality with control rods and in the case of non-boiling water reactors, disolvable boron. In these scenarios neutrons which could have contributed to power production are lost to control elements. In most, if not all, MSR designs the reactors are designed to be operated with almost zero excess reactivity, having just enough fissionable material to stay critical for a small period of time. Either continually or in batches uranium salt may then be added to the flowing fuel salt of the reactor under operation as these fuel lines are near atmospheric pressure and can be accessed. In this way a MSR can be kept critical by the operator with fewer neutrons lost to control elements.

An emerging goal, in terms of operational importance, is reactor safeguards or the measures put in place to prevent illicit diversion of nuclear materials. The traditional approaches to safeguards, namely inspecting and registering fuel assemblies before they go in the core and after as well as through their storage life, don't even apply to MSRs - there are no fuel assemblies and fuel is added in small amounts continuously straight to the fuel line meaning that fissile material is continually accessed, used, and moved. This makes accountancy of such material incredibly difficult. Current efforts are examining if there are any operational characteristics of MSRs which would provide an early indication to plant operators such as a change in the delayed neutron fraction in the core or perhaps a change in the chemical redox potential among other effects. As such, in simulations the ability to model and assess

the impacts of a small diversion of material is necessary.

These considerations here form the general operational constraints of a MSR. A method for simulating the fuel cycle of a MSR must account for all these considerations as well as for the physical phenomena which act on MSRs uniquely in reference to their solid-fuel counterparts.

1.2 Previous Efforts

Various approaches have been made to create a methodology for simulating MSR fuel cycles. Given the span of decades which separate the two major epochs of MSR development there exist two distinct groups of methodologies. One, devised in the 50s and 60s uses numerous approximation and simplifications and was designed in a day when 1 MB of memory was what a supercomputer could give you. While these methodologies may inspire the works of today the actual products of such are lost to time and the confinement of vacuum tube computers to museums. The second was developed beginning in the late 90s with more recent efforts in the 2010s to further expand the computational base employed by these methodologies. A handful of authors of late have proposed approaches for simulating MSR fuel cycles. The differences between these approaches tend to fall into three categories; treatment of the multiplication factor control, assumptions regarding fuel chemistry, and the set of possible reactor operations.

In one of the most influential MSR fuel cycle papers to be published this century, Aufiero unveils a custom modification to the SERPENT 2 code designed to assist in the modelling of the European MSFR project [4]. As all developers must Aufiero made certain assumptions in the development of his modification. For instance, he approximates nuclear criticality as only being dependent on two isotopes, chosen by the user which can be fed and removed proportionally to the deviation from criticality. To handle salt species considerations Aufiero ignores them, simply removing lithium for each fission product produced and adding it for each fission product removed as seen in equation 1.1 where ϕ is the scalar neutron flux,N_j is the number density of isotope j, b is the branching ratio from isotope j into Li, $\sigma_{j\to Li}$ is the transformation cross section for isotope j into lithium, $\sigma_{Li\to j}$ is the transformation cross section for lithium into isotope j, λ_j is the decay constant for isotope j, σ_{kf} is the fission cross section for heavy metal isotope k, and $FY_{k\to l}$ is the fission fragment branching ration for heavy metal isotope k into isotope l [4].

$$\frac{\partial N_{Li}}{\partial t} = \sum_j N_j \phi \sigma_{j\to Li} - \sum_j N_{li} \phi \sigma_{Li\to j} + \sum_j N_j \lambda_j b_{j\to Li} - \sum_{k=HM} N_j \phi \sigma_{kf} \left(\sum_{l=FP} FY_{k\to l} \right) \quad (1.1)$$

As will be seen in chapter 3 this assumption does not always hold given the corrosion concerns of operating a MSR - concerns which Aufiero ignores entirely. Aufiero's modification allows for the user to re-compile SERPENT 2 to make modifications to the nuclear decay

constant of any isotope - in effect creating a proportional feed or removal stream. While Aufiero's work was the first in the modern millennium to demonstrate the reactivity following aspects of a MSR, his modifications to SERPENT 2 proved clunky and inadequate for further pursuit.

In a scripted package Ridley implements a MSR burnup method using SERPENT 2 wrapped with Python [5]. While Aufiero makes the hard assumption that lithium is removed for fission products Ridley goes the other direction and makes the hard assumption that lithium is added for fission products - the authors disagreeing on the oxidative or reducing potential of fission. If fission is on net oxidizing, then a reducing agent such as lithium should be added to the salt as fission progresses. If fission is on net reducing, then an oxidizing agent such as fluorine should be added to the salt as fission progresses.

Other than injecting lithium Ridley largely ignores chemistry and corrosion concerns much like Aufiero. Again, like Aufireo, Ridley instead focuses on finding the fuel feed rate to keep the simulated reactor critical. In his method, at every burnup step, Ridley runs dozens of Monte Carlo core simulations to estimate the impacts of different feed rates. The proposed package then uses Python to fit a polynomial curve to the core reactivity produced by each of the dozens of feed rates. From this curve Ridley's method then selects the "best" feed rate and implements this as a batch addition to the fuel before going through another SEPRENT Monte Carlo and burnup cycle.

Betzler takes a different approach overall by adding wrappers and utilities to the reactor physics package, SCALE [6] [7]. Chemistry control of the simulated system is handled by one of these wrappers. Once TRITON has finished simulating the system of interest from a neutronics perspective the material compositions are passed to this wrapper which adjusts the concentration of specific isotopes to fixed user imposed limits [8]. The resulting feeds and removals which would be needed to achieve these values are then approximated as a proportional removal constant, exactly like Aufiero's work, and then fed to ORIGEN [9]. This wrapper system only has the ability to model proportional flows whose constants do not change over a burnup step - this necessitates rather small, 3 day, burnup steps in order that the proportional constant used does not stray too far from the desired result. With regards to criticality control Betzler employs the crudest method of all those seen, iteratively changing the concentration of a single user specified isotope and re-running the system simulation to see if the desired criticality condition was met. No automated system for addressing chemistry control concerns was included with Betzler's approach.

The approach presented herein consists of a more detailed, customizable, and nuanced solution to the question of MSR burnup. This approach, dubbed ADER for the **A**dvanced **D**epletion **E**xtension for **R**eprocessing, is a source code modification to the popular reactor physics code SERPENT 2 which brings to the user the ability to more accurately model and simulate MSR physics. ADER allows the user, in an abstracted and simplified way, to model chemicals and their interdependent relationships, the impact of material flows on system criticality, the driving factors of corrosion, and the influences of human decisions on a nuclear system - all of this directly integrated into SERPENT 2 with full user support including documentation and a full test suite. In the following chapters ADER is introduced.

First the theory on which ADER is based is presented in chapter 2 along with its integration into the reactor physics Monte-Carlo code SERPENT 2 [10]. In chapter 3 the capabilities of ADER are investigated in relation to a hypothetical MSR fuel cycle. In chapter 4 the concluding remarks on the effects of ADER on MSR fuel cycle modelling are presented as well as recommendations for next steps.

Chapter 2

Method Description and Implementation

In the most direct sense ADER seeks to accomplish two tasks: determining an optimal material composition given a set of constraints, and integrating the necessary composition adjustments into a nuclear material evolution model. The constraints are provided by a user and are built in a generic and system-agnostic manner. Constraints come in three types, material abundance constraints, nuclear constraints, and corrosion constraints. Material abundance constraints are built using the group structures and involve limits on the absolute and relative abundance of groups within the material: e.g. a limit of 4.95 % of the material atomic density for ^{235}U. Corrosion constraints are built around a loose approximation of the Nernst equation as detailed in section 2.2.3. Nuclear constraints are composed of minimum and maximum bounds for the system neutron multiplication factor, k_{eff}, as such ensuring that mass flows into and out of the system do not drive the system away from the desired criticality state.

2.1 Governing Equations

Material abundance constraints are built around the concept of a *group*. A group is a list of elements and their relative proportions; these elements themselves may or may not have specified isotopic proportions. A group, being a list of relative proportions, is arbitrary in size and can be applied to any volume.

From the framework provided by the concept of a group a system of linear equations and relationships can be built which, when taken together, provide an approximation of the constraints in the system to be simulated. Such constraints could include limits on the fractional abundance of a group within a material, bounds on the relative abundance between groups within a material, the sources, sinks, and compositions of mass transfers within the system, just to point out a few. Additionally, applying linear constraints to various weighted sums allows for restrictions on approximations of material qualities such as

the redox potential and the neutron multiplication factor.
Ensuring that the problem constraints and optimization targets are all linear strikes a compromise between the ease and reproducibility and speed of the optimization solution and the model's adherence to the physical phenomenon being simulated. As such care was taken to preserve the linear nature of the material optimization problem. In the following sections the details pertaining to each type of constraint are presented.

2.1.1 Composition constraints

A material, in this framework, is a collection of isotopes. In a nuclear system, it might be desirable to control a material's composition by imposing specific constraints such as chemical solubility and isotopic enrichment limits. These limits are described through the group structure. A single value for the fractional abundance of a group in a material, for example the amount of trifluoride compounds in a fluoride salt, is represented by Equations 2.1 and 2.2

$$g_n = \sum_j^J E_j^{f,n} \tag{2.1}$$

$$g_n = \sum_k^K I_k^{f,n} \tag{2.2}$$

where g_n is the fractional abundance of group n in the host material h, $E_j^{f,n}$ is the fractional abundance of element j in group n - this elemental mass having come from the material to which group n is associated with, h. The f superscript denotes that this is the "future" value of the element's material fraction, or the desired value to which the material streams will seek to move this element to. $I_k^{f,n}$ is the fractional abundance of isotope k in group n - this isotopic mass having come from the material, h, to which group n is assigned. If the constraint is not a single value but a range then Equations 2.1 and 2.2 become the inequalities seen in Equations 2.3 and 2.4 where b_m and b_M are the lower and upper bounds, respectively.

$$b_m \leq \sum_j^J E_j^{f,n} \leq b_M \tag{2.3}$$

$$b_m \leq \sum_k^K I_k^{f,n} \leq b_M \tag{2.4}$$

The second principle constraint involves the relative abundance limits between pairs of groups; a constraint which can be used to approximate chemical solubility limits. Relative group abundance limits can be expressed as seen in Equation 2.5 where $r_{m/M}$ indicates a relative abundance minimum and maximum bound, respectively.

$$r_m \leq \frac{g_1}{g_2} \leq r_M \tag{2.5}$$

Equation 2.5 can be expressed linearly as two inequalities as shown in Equations B.22 and B.23.

$$-\infty \leq -g_1 + r_m g_2 \leq 0 \tag{2.6}$$

$$0 \leq -g_1 + r_M g_2 \leq \infty \tag{2.7}$$

2.1.2 Material flow constraints

Another set of constraints regards the sources, sinks, and compositions of mass flows to, from, and between materials. These constraints are applied through the *stream* structure; the definition of which can be seen in Equations 2.8 and 2.9.

$$s_l = \sum_j^J E_j^{d,l} \tag{2.8}$$

$$s_l = \sum_k^K I_k^{d,l} \tag{2.9}$$

Taking s_l to be stream l, $E_j^{d,l}$ to be the change in the abundance, as indicated by the superscript d, of element j in the affected material, h, as caused by stream l. $I_k^{d,l}$ is taken to be the change in the abundance of isotope k in the affected material, h, as caused by stream l. These equations describe that mass flows into, out of, and between materials are composed of elements and that this elemental composition must be matched by the isotopic composition of the mass flows. Such detailed accounting of entities is required as any linear optimization solver is unaware that elements comprise their constituent isotopes

2.1.3 Oxidation constraints

A weighted sum over all elements in a material with minimum and maximum target values, O_m and O_M respectively, forms the next constraint which can be applied to a material - equation 2.10. Although the framework for this summation was implemented as a rough approximation to redox potential monitoring in liquid systems (discussed more in section 2.2.3) there is no reason the weights involved could not represent some other quantity of interest to the user.

$$O_m \leq \sum_j^J w_{E_j} E_j \leq O_M \tag{2.10}$$

where w_{E_j} is a weighting factor which can be applied to any element and which exists solely for the convince of the user.

2.1.4 Reactivity constraints

A weighted sum over isotopes in a material forms the reactivity constraint that may be applied. This constraint is derived from the expression for the multiplication factor as found in Equation B.32:

$$k_{eff} = P_{NL} \frac{\sum_{m}^{M} \phi_m \omega_m \nu\Sigma_f^m}{\sum_{m}^{M} \phi_m \omega_m \Sigma_a^m} \tag{2.11}$$

where ϕ_m, ω_m, $\nu\Sigma_f^m$ and Σ_a^m are, respectively, the scalar neutron flux, the volume fraction, the spectrum averaged neutron production cross section, and the macroscopic absorption cross section for each material m. P_{NL} is the neutron non-leakage probability. In ADER, the ability to control k_{eff} is limited to the case of a single neutron multiplying material; take $\nu\Sigma_f = 0$ for every material but the multiplying material H and as such Equation B.32 can be rewritten as follows:

$$k_{eff} = P_{NL} \frac{\phi_H \omega_H \Sigma_a^H}{\sum_{m}^{M} \phi_m \omega_m \Sigma_a^m} \frac{\nu\Sigma_f^M}{\Sigma_a^M} \tag{2.12}$$

The probability of a neutron being absorbed in the multiplying material is defined as follows:

$$P_A = P_{NL} \frac{\phi_H \omega_H \Sigma_a^H}{\sum_{m}^{M-1} \phi_m \omega_m \Sigma_a^m} \tag{2.13}$$

Then k_{eff} can be calculated as:

$$k_{eff} = P_A \frac{\nu\Sigma_f^H}{\Sigma_a^H} = P_A \frac{\sum_{k}^{K} \nu\Sigma_f^k}{\sum_{k}^{K} \Sigma_a^k} \tag{2.14}$$

where $\nu\Sigma_f^k$, and Σ_a^k are, respectively, the spectrum averaged neutron production cross section, and the absorption cross section for every isotope k in the multiplying material H. This relation is expected to hold for simulations in which there is a dominant reactive material and for which $\nu\Sigma_f \approx 0$ for all other materials.

In this case, given lower and upper bounds for the multiplication factor of the system, k_{eff}^{min} and k_{eff}^{max} respectively, Equation B.35 can be made linear as in Equations B.36 and B.37.

$$0 \geq \frac{k_{eff}^{min}}{P_A} \sum_{k}^{K} \sigma_a^k I_k - \sum_{k}^{K} \nu^k \sigma_f^k I_k \tag{2.15}$$

$$0 \leq \frac{k_{eff}^{max}}{P_A} \sum_{k}^{K} \sigma_a^k I_k - \sum_{k}^{K} \nu^k \sigma_f^k I_k \tag{2.16}$$

A key assumption of this linearization process is that $\frac{\partial P_A(m...M)}{\partial M} = 0$ when in truth P_A, the leakage probability, is a function of the composition of material m and each other material in the simulation. The impacts of this approximation are expected to be quite small as most nuclear simulations have a core of critical material whose reactivity and leakage are very loosely coupled with the material around them. However it will affect all simulations, more so those with strong leakage effects and neutronic feedback from neighboring materials, such as simulations of systems dependent on reflectors.

2.1.5 The optimization method

Equations 2.3, 2.4, B.22, B.23, 2.8, 2.9, 2.10, B.36, and B.37 demonstrate the linearity of the problem. Given a linear set of equations, and a set which in various configurations could be under-constrained, over-constrained, or equal, a unique solution can not be guaranteed at all times. As such the best 'solution' is an optimization route through which the addition of an optimization target can guarantee a unique solution. With respect to the the ease of implementation, the ease of use, and the comprehensibility of the simulation results, linear optimization or linear programming as it is sometimes known, was chosen as the optimization method. The linear programming problem is represented by a sparse matrix which is manipulated to produce the optimal solution given an optimization target (details in section 2.2.6).

2.2 Implementation in SERPENT 2

Utilizing the linear relationships described in section 2.1, ADER brings to SERPENT 2 the ability for users to define desired relationships between constituent units of a material and to define material flows within the system. Additionally ADER provides tools for constraints based upon the elemental composition of a material as well as constraints relating to the multiplication factor of a system. In this section the realization of these tools in SERPENT 2 is detailed.

2.2.1 Groups

A *group* is an elementary structure in the ADER framework as described in section 2.1.1. Elaborating further, a group is composed of a fixed set of elements, with or without specified isotopics, with fixed abundances relative to the group as a whole, e.g., a group could be made to specify that it is one part uranium and four parts chlorine (uranium tetrachloride). Furthermore, the uranium could be specified to be 4.95% ^{235}U and 95.05% ^{238}U. A group is not a *material* in the SERPENT meaning of it, but rather a material constituent that is connected to a material by set relations. For example, the user can define the material FLiBe as 2LiF-BeF_2, then define two groups as LiF and BeF_2 and then specify the constraint that the material FLiBe maintains a 2:1 ratio between the two groups regardless of any other occurring change. As stated in section 2.2 a group can be applied to any control volume and itself has no inherent density associated with it.

ADER provides several means to define the relationships between groups, and between materials and groups. To define relationships between groups a range of relative abundances between any two groups in the same material may be defined for any arbitrary number of group pairs. To define relationships between the material and the related groups, a range of absolute abundances of a group in a material may be specified for an arbitrary number of groups. The governing equations of these relationships are found in section 2.1.1. To facilitate modelling of chemical compounds with related and possibly interchangeable forms, a group in ADER may also be formed from the linear combination of any other groups previously defined. These three simple mechanisms can be combined to model various chemical situations and form a core component of the conditions for optimality placed on a material.

Say, for example, that a material is desired to have three to four times as much eutectic FLiBe salt to uranium fluoride salts, both uranium trifluoride and tetrafluoride. Additionally, it is desired that uranium tetrafluoride be more than 100 times as abundant as uranium trifluoride. Setting these as constraints for the material can be accomplished with four groups and two relative abundance constraints. The following groups are needed: a uranium trifluoride group, a uranium tetrafluoride group, a uranium fluoride group obtained as a summation of the previous groups, and a FLiBe group. A relative abundance constraint is placed between the FLiBe group and the uranium fluoride group, and a relative constraint is placed between the two uranium fluoride salt groups. With those six constructs a solubility constraint on a family of related compounds is put into effect. This is just one example of many restraints and conditions which can be modeled with the ADER group structures and the relationships between them.

Finally, related to the group structure in ADER, is the concept of *free* versus *controlled* elements or isotopes. In ADER, for a given material, whether or not an element or isotope should be completely accounted for by the groups which possess these constituents or be allowed to have free portions not locked up in the group structure is something that can be specified. For example, if a material were to have a uranium tetrafluoride group the fluorine in that material would be *controlled* when all the fluorine in that material were required to

be accompanied by 0.25 uranium atoms. If the fluorine content of the material were left to be *free* then the material would be allowed to have a fluorine to uranium ratio less than 0.25—indicating that not all fluorine in the material is bound in a UF_4 compound.

2.2.2 Streams

A *stream* is both the workhorse and the end-goal of ADER. There are two classes of streams in ADER: group-class and table-class. Group-class streams are options. They represent pathways available to ADER to move mass into, out of, and between SERPENT materials with the goal of bringing their compositions to an optimal state. Table-class streams are prescriptive. They are directions to ADER to move specific types and amounts of mass from and to specific materials —the results of table-class streams are factored into the material composition before determination of optimality allowing the effects of group-class streams to reflect the consequences of table-class stream effects. All streams have a set of common attributes provided by the user: a source, a sink, and the behavior in time of the stream. For the majority of streams a source and a sink are optional, but at least one of the two must be provided. Missing sources are treated as infinite supplies of whatever substance is needed; missing sinks are treated much like sinks—endless consumers of disposed mass. In terms of their behavior in time, ADER supports three types of streams: discrete type stream transfers happen between burnup steps as step changes; continuous type stream transfers occur as a steady rate of mass transfer over the length of a burnup step; proportional type streams modify the decay constant of isotopes, even to the point of making the decay constant a production constant if that is what is called for. A notable feature related to streams is the option to require that inflows match outflows for specified materials; this dramatically simplifies the set of constraints needed to model a practical system in which the mass is not simply allowed to vary between 0 and ∞.

Group-class streams have an additional attribute the user is required to set: the ADER group which defines the substance the stream will move. These streams are given no set amount of mass transfer. Rather, through its optimization process ADER determines the amount of mass transfer each group-class stream should have. Table-class streams have two additional attributes the user is required to set: the ADER transfer table to be used and a positive value, denoted c^s. Transfer tables in ADER are user defined lists of selected elements and isotopes all of which have some value attached to them, denoted c_k^t. Multiplying the value from the transfer table with the value given in the table-class stream definition gives the fraction of the whole for an individual isotope or element that will be moved by the table-class stream per unit time over the next burnup step. For proportional type streams the value produced by this multiplication will be added to the decay constant of the appropriate isotopes, or subtracted given the stream's relation to the material in question.The value of splitting the table-class stream mass transfer rates into two numbers lies with MSR modelling. In many proposed MSR designs there is some fuel treatment procedure which is applied to some fraction of the fuel salt, represented by the value given in the table-class stream definition. This treatment procedure removes specific elements with differing

effectiveness, as represented by the value given to each element and isotope contained in a transfer table.

Streams, group-class or table-class, have clear applicability to MSR modelling. ADER's optimization routines, in this instance, should be thought of as the reactor operator with the streams representing those mass flows in and out of a reactor that the operator may plan; such as an addition of lithium fluoride for maintaining a desired salt condition or the addition of ^{233}U for criticality control. Table-class streams provide a means not only to model possible fuel salt reprocessing options but also natural process which change the composition of fuel salts such as the escape of noble gas fission products. Outside of MSR modelling streams find other applications ranging from geological repository modelling to biological radiation dose analysis.

2.2.3 Oxidation control

As mentioned in section 2.1.3 the oxidation control portion of ADER is a weighted sum over the elements in a material with bounds for the evaluation of the sum set by the user. In the oxidation table structure a complete list of elements with their expected average oxidation state, or whichever weighting factor is used, in the desired material is given. This structure was designed with MSR operations in mind. The redox potential of a flowing liquid is a key parameter of interest in many liquid fuel reactor designs. It is far beyond the scope of ADER, and even SERPENT 2, to be determining the redox potential of a chemical mixture through simulation. That said, it is suspected that this feature will greatly ease the burden on most MSR simulations as most proposed fuel salts have a dominant anion which bonds with near everything else to the exclusion of other compounds. Combining this tool with the principles from the Nernst equation, Equation 2.17, bounds for the average redox potential of a molten salt can be set—a key metric in controlling corrosion in molten salt systems. In Equation 2.17 ϵ_i is the redox potential, ϵ_i^o is the redox potential in the standard state, R is the gas constant, T is the temperature, z is the number of electrons received by the oxidizing agent, j is Faraday's constant, $[oxid]$ is the activity of the oxidized species while $[red]$ is the activity of the reduced species. Conversions between activity and concentration are required but approximations may be sufficient; this task is left to the user. These conversions render the expected oxidation state for all constituents of the Nernst equation as activity is a partial function of oxidation state.

$$\epsilon_i = \epsilon_i^o + \frac{RT}{z_\mathrm{J}} \ln\left(\frac{[oxid]}{[red]}\right) \tag{2.17}$$

2.2.4 Reactivity control

The multiplication factor of a nuclear system is regularly of interest and often desired, in a reactor, to be of a specific value. As such users may set system wide k-eigenvalue constraints through ADER. When such constraints are applied ADER incorporates Equations B.36 and

B.37 into the linear optimization problem so that the effects of mass transfers within the system can be constrained by their predicted impacts on the multiplication factor of the system. The relevant cross section information, for all isotopes in an ADER material is retrieved from SERPENT 2 and used to fill in Equations B.36 and B.37.

Even if Equation B.32 is exact, k_{eff} is only ever approximated; not just from the error inherent in Monte Carlo simulations, but from ignoring that every term in Equation B.32 is non-linearly dependent on composition. As such it is expected that the reactivity control feature of ADER will only behave well for small changes in composition that do not have a large effect on the neutron flux. Finally, more of a limitation than an assumption, ADER has no means to measure the effect on reactivity resulting from changes in one material interacting neutronincally with another. As such ADER's reactivity control feature only works in situations where one material is the dominant driver of criticality in a system. Partially to address these shortcomings, ADER offers the user the option of setting the number of max iterations allowed per burnup step to determine the reactivity effects of ADER's streams. At the end of each such iteration a Monte Carlo cross section calculation is repeated to assess the effects of ADER's actions.

2.2.5 The optimization target

Having covered the many constraints available to be placed on the model, the missing piece to the linear optimization problem is an optimization target. To this end ADER allows the user to set the optimization direction, minimization or maximization, as well as the optimization target from which the user can select such options, as a specific group in a specific material, a specific stream, all material transfers, specific material transfers, and others detailed in the ADER user manual in appendix B.

2.2.6 The optimization matrix

The CLP library expects a linear programming matrix from ADER—one built from all the constituent equations in the ADER scheme. Before the equations presented earlier in this section can be incorporated into the matrix a note about the effects of table-class streams should be made. As mentioned in section 2.2.2 table-class streams are prescriptions, meaning the mass flows they stipulate are going to happen. This information makes it into the linear programming matrix in the form of adjustments to the bounds of specific rows. These adjustments are denoted as r_x where r is the net positive increase in the abundance of component x as caused by all table-class streams. It should be noted that adjustments calculated for proportional removal table-class streams are only approximations, and sometimes poor approximations, of the actual amount of an isotope or element that will be removed as nuclear processes change the abundance of isotopes in a way that ADER is currently unaware of.

Figure 2.0 depicts the scheme for constructing the linear programming matrix. Column bounds , seen above the label describing what the column represents, are given as are row

bounds which are seen to the left of the label describing which equation the row represents. For the sake of brevity the matrix in figure 2.0 is for one material only though many materials may be involved in such a matrix should they be linked together by shared mass transfers. In which case the only variables shared between materials are the group-class streams and the stream equations they are a part of are the only coupling equations; aside from transfers by table-class streams but those are only represented in the linear programming matrix, they are handled by other routines all together. If a second material were to be included in this matrix then, perhaps, the stream entries in the third and fourth columns would have non-zero coefficients for some E_j^d and I_k^d rows of the second material.

Working down the matrix row by row the first row encountered represents Equation B.22 with arbitrary groups g_1 and g_2 whereas the next row down represents Equation B.23. The third row, what will be referred to as an elemental future row, represents the atom balance for element j where $f^n_{E_j^f}$ is the fractional proportion of element j in group n. The novel column involved here is an elemental future column whose inclusion in the same row closes the equation. The bounds for this row are those for a free element or those elements which are permitted to have portions of the element not tied up in declared group structures. In the case of a controlled element, those whose complete abundance must be accounted for by group structures, the lower bound is changed to zero. The fourth row, an elemental delta row, represents the change in the abundance of element j as caused by all group-class streams where $f^n_{E_j^f}$ is the fractional proportion of element j in stream n. Of course the elemental delta column is involved to close the balance. The fifth row, or balance row, is what ties together E_j^f and E_j^d. The bounds, α and β, are equal and represent $E_j^c + r_{E_j^f}$ constituting an atom balance "in time" where E_j^c represents the present fractional abundance of element j. The fifth row requires, straightforwardly, that the future amount of an element be equal to the current amount plus any delta, or change, in the element's abundance. The sixth row is an isotopic balance row requiring that the abundance of an element be equal to the abundance of its constituent isotopes. The following three rows, the seventh, eighth, and ninth, are the isotopic versions of the elemental future, delta, and balance rows where f values are for the isotopic fractional proportions. γ and δ are equal and represent $I_k^c + r_{I_k^c}$ where I_k^c represents the present fractional abundance of isotope k. In the tenth and eleventh rows Equations B.37 and B.36 find representation with η and θ respectively representing terms of the expanded sum found in the referenced equations; $\frac{k_{eff}^{min}}{P_A}\sigma_a^k - \nu^k\sigma_f^k$ and $\frac{k_{eff}^{max}}{P_A}\sigma_a^k - \nu^k\sigma_f^k$. The twelfth row represents Equation 2.10, accounting for the contributions of the future quantity of elements to a material's averaged oxidation state. The thirteenth row, or Pres row, exists when the user instructs ADER to balance inflows with outflows. The pres row requires that the net stream transfers in a material come to zero. The effects of table-class streams are captured in υ and ω as seen in Equation B.38 where s^t is a table class stream abundance value. The final row is the optimization, or Opt row. This row indicates to the simplex routine which variables to minimize or maximize. In figure 2.0 the opt row is indicating that g_1 is the optimization target. The direction of optimization, maximization or minimization,

is a parameter which the user passes.

		$[b_m, b_M]$	$[0,\infty)$	$[0,\infty)$	$[0,\infty)$	$[0,\infty)$	$(-\infty,\infty)$	$[0,\infty)$	$(-\infty,\infty)$
		g_1	g_2	s_1	s_2	E_j^f	E_j^d	I_k^f	I_k^d
$(-\infty,0]$	Eq.B.22	-1	r_m						
$[0,\infty)$	Eq.B.23	-1	r_M						
$(-\infty,0]$	E_j^f	$f^1_{E_j^f}$	$f^2_{E_j^f}$			-1			
$[0,0]$	E_j^d			$f^1_{E_j^f}$	$f^2_{E_j^f}$		-1		
$[\alpha,\beta]$	E_j^b					1	-1		
$[0,0]$	E_j^i					-1		1	
$(-\infty,0]$	I_k^f	$f^1_{I_k^f}$	$f^2_{I_k^f}$					-1	
$[0,0]$	I_k^d			$f^1_{I_k^f}$	$f^2_{I_k^f}$				-1
$[\gamma,\delta]$	I_k^b							1	-1
$[0,\infty)$	Eq.B.37							η	
$(-\infty,0]$	Eq.B.36							θ	
$[O_m, O_M]$	Eq.2.10					$o_{E_j^f}$			
$[\upsilon,\omega]$	Pres			1	1				
	Opt	1							

Figure 2.0: A depiction of the Simplex optimization matrix.

$$\upsilon = \omega = -\sum_{s'}^{S'} s_{s'}^t \tag{2.18}$$

2.2.7 Solution and limitations of the linear optimization problem

To solve the linear programming problem ADER employs the CLP library from the COIN-OR project [11]. CLP is a double-precision linear optimization solver utilizing the Simplex algorithm. Sandia National Laboratory noted in their report on open-source linear solvers that CLP was by far the fastest, most accurate, and most capable of the solvers they tested and that it performed on the same order of magnitude for any metric when compared against commercial solvers [12]. Once ADER has constructed the sparse matrix representing the linear optimization problem and packed this matrix into a dense column major format said matrix is handed off to the CLP simplex solution routines. CLP solves the linear programming problem and returns back a vector containing the value of the objective function as well as the values all the variables take in the optimal solution. The key pieces of information from this process are the values of the stream abundances. However, these stream abundance values are burdened by two key limitations both related to the time-independence of the linear optimization problem.

The first limitation can be entirely avoided if all streams in a given simulation have identical behavior in time. If streams with differing behavior in time affect a common material the optimization solution may not be true at any or all points in time for that solution interval. In a simple example, imagine that material A requires four parts fluorine from stream B and one part uranium from stream C. Take stream B to be a discrete type stream and stream C to be a continuous type stream. Neglecting any nuclear depletion the optimal composition will not be realized until the end of the burnup step when stream C has delivered all of its uranium. Additionally, a previously unexpected amount of fluoride without any corresponding uranium will appear in the material suddenly when the effects of stream B are applied. Simulations involving streams with mixed time behavior may find recourse with shorter burnup steps but mixing streams with differing time behavior is ill-advised in general.

The second limitation arises from the effects of nuclear depletion. Should nuclear depletion act upon a constituent of the linear optimization solution the solution may not hold following the effects of nuclear depletion. This limitation will be most strongly felt in simulations for which an isotope with a large rate of change in its concentration is also a key constituent of an optimization problem, particularly one with less flexible constraints. Shorter burnup steps which minimize the integrated effect of an isotope's rate of change may reduce the degree to which the actual composition diverges from the ideal composition.

The greatest limitation of the solution turns out not to be related to the physics of the problem - but to the floating point precision of computers. As put forward in [13] the limitations of machine precision become critical in SIMPLEX algorithms written with 64 bits or less of precision and with variables having relative magnitudes at or lower than 10^{-6}. This issue is easily resolved by employing a SIMPLEX solver which uses quadruple or higher precision for floating point numbers.

The impact of this limitation is difficult to understate or pin down. Whether or not a small number will cause the failure of a SIMPLEX solution depends on the remainder of the coefficients in the optimization problem and how the SIMPLEX algorithm goes about finding the solution. In use-testing many simulations of postulated nuclear systems were able to be run with ADER out to thousands of years of effective reactor operation. In many more cases the simulations fail to find a SIMPLEX solution and exit before a year of effective reactor operation has passed.

Any linear transformation applied to the optimization problem preserves the overall problem of $x >> y$ where x and y are given values in the optimization problem. Furthermore, in many nuclear simulations several isotopes have significant importance to the purpose of the simulations and yet routinely have atomic densities below 10^{-6} relative to their host material atomic density - ^{6}Li being a common example.

Considering the limitations of machine precision on the efficacy of ADER, until a SIMPLEX algorithm employing quadruple or higher precision for floating point arithmetic is incorporated ADER must be considered stochastically functional — certainly a less than optimal state. While the authors of [13] provide their quad-precision code, it is written in FORTRAN which would necessitate the construction of a wrapper function. Additionally

their code takes a much more detailed and error prone format than does CLP - as such necessitating not insignificant structural modifications to ADER. A significant development effort would be needed to either implement the FORTRAN library or to update the CLP library to quad-precision.

Were this development effort to be undertaken the complexity of the task would be minimal. There is one function *ADEROperateMaterialCompMatrix* which would need to be modified. The functions *ADERBuildClpModel* and *ADERSolveClpModel* would need to be replaced with appropriate functions for transmitting the sparse matrix in ADER's memory to the quad-precision library of choice and calling said solution.

2.2.8 Material depletion

Following the solution of the optimization problem discrete type streams have their effects applied before the burnup step begins. A Monte Carlo simulation is then run and if the multiplication factor is outside of the user defined bounds (and iterations remain, as set by the user) another optimization solve will be executed except the changes already made by discrete type streams remain. Beyond this, ADER modifies the burnup matrix inside of SERPENT 2 to reflect the effects of streams. The coefficients in the burnup matrix are those in the Bateman equation as seen in Equation B.39 for one energy group and zero dimensional case, where N is the number density of nuclide n, t is time, $b_{m\to n}$ is the branching ratio for the decay of nuclide m into n, λ is the decay constant for its sub-scripted nuclide, q goes over all neutron induced absorption reactions for a given isotope, $a^q_{m\to n}$ is the branching ratio for isotope m into n due to reaction q, σ^y_x is the effective microscopic cross section of reaction x for isotope y, ϕ is the scalar neutron flux, d denotes all transmutation reactions for a given isotope, $R_n(t)$ is a fractional removal (or addition) rate for isotope n at time t, and $F_n(t)$ is a feed (or removal) amount for isotope n at time t.

A highly truncated burnup scheme can be seen in figure B.40 in which there are two isotopes, ^{233}U and ^{135}Xe, and two streams; S_c representing a continuous stream with a constant injection rate and S_p representing a proportional stream with a transfer rate dependent upon the concentration of the substances to be transferred. There are, of course, two matrices as well. The burnup matrix to the left holding the coefficients of the Bateman equation and the second, to the right, holding the initial concentrations of isotopes and the values for the streams. The first column of the first row gives the creation and destruction of ^{233}U which is dependant on the concentration of ^{233}U with Γ representing nuclear destruction as seen in equation B.41. The third column of the first row holds the fraction of stream S_c that ^{233}U comprises. These entries together describe the evolution of ^{233}U in the given system. In the second row Ξ, as seen in Equation B.42, represents the production of ^{135}Xe from ^{233}U. In the second column of the second row are the processes dependant on the concentration of ^{135}Xe. Υ represents the proportional rate constant as determined by the multiplication of $c^t_{^{135}\mathrm{Xe}}$ and c^s whereas Θ is given by Equation B.43. The third row is blank as the abundance of a continuous type stream, h_{S_c}, does not change over a burn step. The fourth row is an addition specific to ADER and not found in the Bateman equations; rather, this line, and the lines it

represents, exists to keep track of the amount of an isotope that a proportional stream moves simply to provide this information to the user. The system of matrices seen in figure B.40 is solved by SERPENT 2 providing updated isotopic abundances and proportional stream transfer amounts.

$$\begin{aligned}\frac{\mathrm{d}N_n(t)}{\mathrm{d}t} =& \sum_m^M b_{m\to n}\lambda_j N_j(t) + \\ & \sum_m^M \sum_q^Q a^q_{k\to i}\sigma^k_q \phi(t) N_k(t) - \\ & N_n(t)\lambda_i - \sum_d^D \sigma^n_d \phi(t) N_n(t) - \\ & R_n(t)N_n(t) + F_n(t)\end{aligned} \tag{2.19}$$

$$\begin{array}{c} \\ {}^{233}\mathrm{U} \\ {}^{135}\mathrm{Xe} \\ S_c \\ S_p \end{array} \begin{array}{c} \begin{array}{ccccc} {}^{233}\mathrm{U} & & {}^{135}\mathrm{Xe} & S_c & S_p \end{array} \\ \begin{bmatrix} -\lambda_{^{233}\mathrm{U}} + \Gamma & & f^{S_c}_{^{233}\mathrm{U}} & \\ \Xi & -\lambda_{^{135}\mathrm{Xe}} + \Upsilon + \Theta & & \\ & & & \\ & \Upsilon & & \end{bmatrix} \end{array} \begin{array}{c} \mathbb{N} \\ \begin{bmatrix} N_{^{233}\mathrm{U}} \\ N_{^{135}\mathrm{Xe}} \\ h_{S_c} \\ 0 \end{bmatrix} \end{array} \tag{2.20}$$

$$\Gamma = -\sum_d^D \sigma_d^{^{233}\mathrm{U}} \phi \tag{2.21}$$

$$\Xi = b_{^{233}\mathrm{U}\to{}^{135}\mathrm{Xe}}\lambda_{^{233}\mathrm{U}} + \sum_q^Q a_{^{233}\mathrm{U}\to{}^{135}\mathrm{Xe}}\sigma_q^{^{233}\mathrm{U}}\phi \tag{2.22}$$

$$\Theta = -\sum_d^D \sigma_d^{^{135}\mathrm{Xe}} \phi \tag{2.23}$$

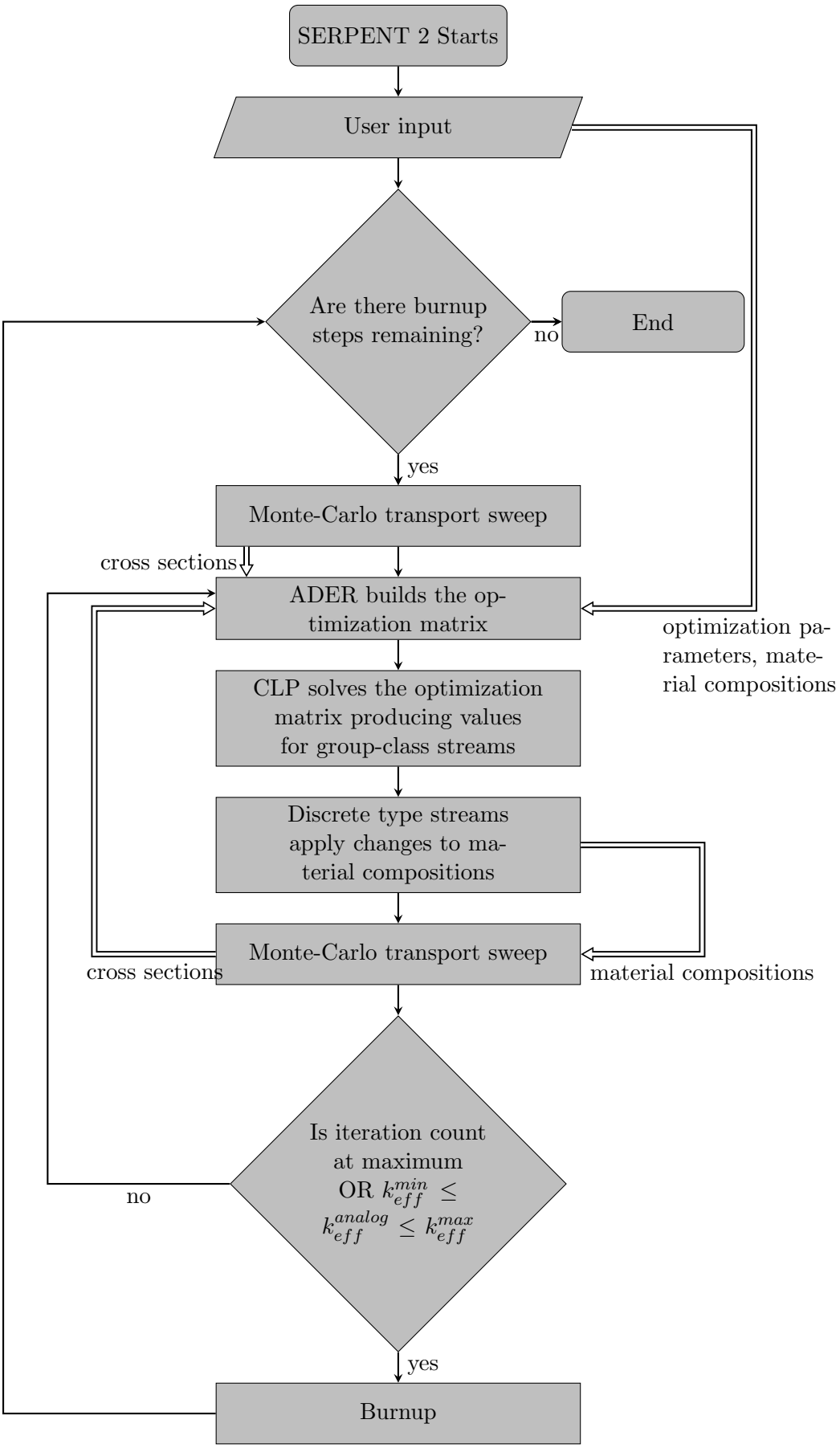
2.2.8.1 Iterations

The Bateman equation Eq.B.39 is not a standalone description of the isotopic evolution of a nuclear system; rather, it is tightly coupled with the scalar neutron flux. Considering that the solution of the system of matrices in figure B.40 does not solve for the scalar neutron flux it is clear that the solution, if for nothing else, is an approximation. Many nuclear material evolution schemes iterate between solutions of the neutron flux and the Bateman equations within the same burnup step—SERPENT 2 is no different. Although the purpose of this section is not to investigate the burnup solution routines of SERPENT 2, those can be seen

in [14], the iteration scheme employed by these routines could affect ADER. In truth, ADER is compatible with any iteration scheme employed by SERPENT 2 consequently in part to a limitation of ADER — due to the mix of continuous and discrete streams convergence of an iteration scheme for optimization involving nuclear processes on the fuel is impossible to guarantee. At the present time ADER only iterates to check the reactivity component of its solution, as mentioned in subsection B.11.7. After the linear programming matrix has been built and solved, the effects of discrete type streams are applied to the pertinent SERPENT 2 materials. Following the application of discrete type streams the transport sweep is re-run. If the system analog k_{eff} is within bounds as set by the user, program-flow will continue on to the building and solution of the burnup matrices; otherwise, ADER re-builds and re-solves the linear programming matrix and applies the new discrete type streams on top of the changes made by the previous iteration of discrete type streams. These iterations happen at the beginning of every burn step in which the Bateman equations will be solved. Other than these actions, ADER does not interact with Serpent 2 burnup iterations schemes. A flowchart roughly outlining SERPENT 2 and ADER interaction is seen in figure 2.1.

2.2.9 Algorithm implementation

Concerning the software engineering aspects of ADER's creation the most useful resources to any curious individual are the API, user manual, and source code provided in appendices A and B respectively while the source code is online along with the system tests described later in this paragraph. ADER was developed in a test-driven environment and as such has more than 150 unit tests supporting its development as found in the file `testcases.c`. More than a dozen integration tests can be found within the code contained in functions which begin with the prefix `TEST`. Lastly, more than 20 system tests can be found online at `www.github.com/ddwooten/ADER_pub/System_Testing/`. ADER's directory structure includes no levels below the `/src` folder as the parent project, SERPENT 2, does not either. ADER's code style adheres closely to that of SERPENT 2 but adopts a more readable use of white space. The input for ADER is directly integrated into the SERPENT 2 input and uses the same style.


SERPENT 2 Starts
User input
Are there burnup steps remaining?
no
End
yes
Monte-Carlo transport sweep
cross sections
ADER builds the optimization matrix
optimization parameters, material compositions
CLP solves the optimization matrix producing values for group-class streams
Discrete type streams apply changes to material compositions
Monte-Carlo transport sweep
cross sections
material compositions
Is iteration count at maximum OR $k_{eff}^{min} \leq k_{eff}^{analog} \leq k_{eff}^{max}$
no
yes
Burnup

Chapter 3

Test Cases and Results

In the following sections a nuclear fuel cycle analysis as carried out by ADER, investigating an infinite and homogeneous mixture of fuel salt, is presented. A fuel salt composed of $LiF-BeF_2-ThF_4-{}^{233}UF_4$ is exposed to a neutron flux such that a power density of 100 Wcm^{-3} is held throughout the simulation. Various constraints are applied through the ADER interface and the effects of these constraints are investigated.

3.1 Simulation Setup

The system configuration is described in this section with corresponding SERPENT 2 input provided in appendix C. In short the simulation consisted of an infinite and homogeneous salt mixture, 71.8 mol-% LiF, 16 mol-% BeF_2, 10.8 mol-% ThF_4, and 1.2 mol-% ${}^{233}UF_4$ - the startup fuel for the Oak Ridge Molten Salt Reactor Experiment [1]. The starting lithium load was enriched to 99.995 % ${}^{7}Li$. The lithium fraction of the homogeneous mixture was constrained to be between 0.26 and 0.30 using a range restriction. The total amount of lithium within the system was tracked using a catch-all elemental group and a total summation group counting both the lithium acting as a primary salt constituent and the lithium which may be free or bound to fission products within the system. Separate groups of elemental fluorine are used in ratio restrictions to impose chemical binding constraints on the Li, Be, Th, and U within the salt mixture. A free elemental group of fluorine as well as a total fluorine summation group are used to track total system fluorine. An elemental beryllium group constrains beryllium to between 6.1 mol-% and 6.5 mol-%. The molar fractions of ThF_4 as well as UF_4 - all uranium is assumed to be uranium-IV - are left free to vary as ADER chooses. An oxidation range of $[-0.0002, -0.0001]$ is used to ensure a reductive environment within the salt with each element's assumed oxidation sate within the salt visually depicted in figure 3.1. Any lithium, beryllium, fluorine, thorium-232, and uranium are required to be represented in a group structure during the linear optimization. The optimization target is set as the minimization of total gross flows.

The stream modelling natural removal removes the following elements into an infinite

sink with a 30 second effective-half-life: He, Ne, Ar, Kr, Nb, Mo, Tc, Ru, Rh, Pd, Ag, Sb, Te, Xe, and Rn. Those elements in the gas phase given MSR operating temperatures and pressures - around one atmosphere and generally between 550 °C and 800 °C - do not remain in the salt and generally bubble out either in a helium gas bubble trap or in the pump bowl. The remaining elements in the natural removal stream are considered the 'noble' metals and will plate out, most commonly, on the cold leg of a flow loop. Previous modelling by [4] has shown that the 30 second effective half-life captures these physical effects well. Figure 3.1 provides a visual depiction of the natural removal stream, the reprocessing stream, as well as the assumed oxidation state of each element in the salt which is seen in the upper right corner of each element's box and for which a number not preceded by a minus sign indicates an oxidized state. All elements, other than the chalcogens and halogens, are assumed to be in their highest oxidation state unless operational data from [1] indicates otherwise.

The stream modelling the reprocessing of the fuel salt removes the following elements into an infinite sink: B, N, C, O, Na, Mg, Al, Si, P, S, Cl, K, Ca, Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Lu, Hf, Ta, W, Re, Os, Ir, Pt, Au, Hg, Tl, Pb, Bi, Po, At, Fr, Ra, Lr, Rf, Db, Sg, Bh, Hs, Mt, Ac, Pa, Zn, Ga, Ge, As, Se, Br, Rb, Sr, Y, Zr, Cd, In, Sn, I, La, Ce, Pr, Nd, Pm, Sm, Eu, Gd, Tb, Dy, Ho, Er, Tm, Yb, Cs, and Ba. This removal occurs such that half of each element's total abundance is removed every 10 years.

The remaining streams in this simulation are all group-class streams whose quantity is determined on a per burnup step basis by the linear optimization routine. All of these remaining streams inject or remove their assigned quantity, per burnup step, of material at a continuous and unchanged rate with respect to time. These remaining streams are a lithium injection stream enriched to 99.995 % ^{7}Li, an elemental LiF removal stream, a ^{9}Be injection stream, an elemental BeF_2 removal stream, a ^{19}F injection stream, an elemental fluorine removal stream, a $^{232}ThF_4$ injection stream, an elemental ThF_4 removal stream, a $^{233}UF_4$ injection stream, and an elemental UF_4 removal stream.

With regards to simulation parameters, the burnup steps were measured in days and grew in length according to the Fibonacci sequence up to 34 days at which each remaining burnup step was kept to 34 days. The minimum and maximum neutron multiplication factor targets were 1.0 and 1.01 respectively. A linear extrapolation and interpolation scheme was selected for the nuclear cross sections used in the standard CRAM burnup calculation. All possible nuclides were tracked. Each Monte-Carlo system simulation was run with 10k neutrons for 40 inactive cycles with 60 active cycles; average cycle error on neutron multiplication was approximately 150 pcm. Simulations were carried out on the Savior HPC cluster here at UC Berkeley on a Dell PowerEdge C6220 server blade equipped with two Intel Xenon 10-core Ivy Bridge processors all running at 2.5 GHz having 64 GB of memory.

3.2 Simulation Outputs

In figures 3.2 through 3.27 the results of the simulation described in section 3.1 - with input given in appendix C - are given below. Following these figures in section 3.3 the lessons

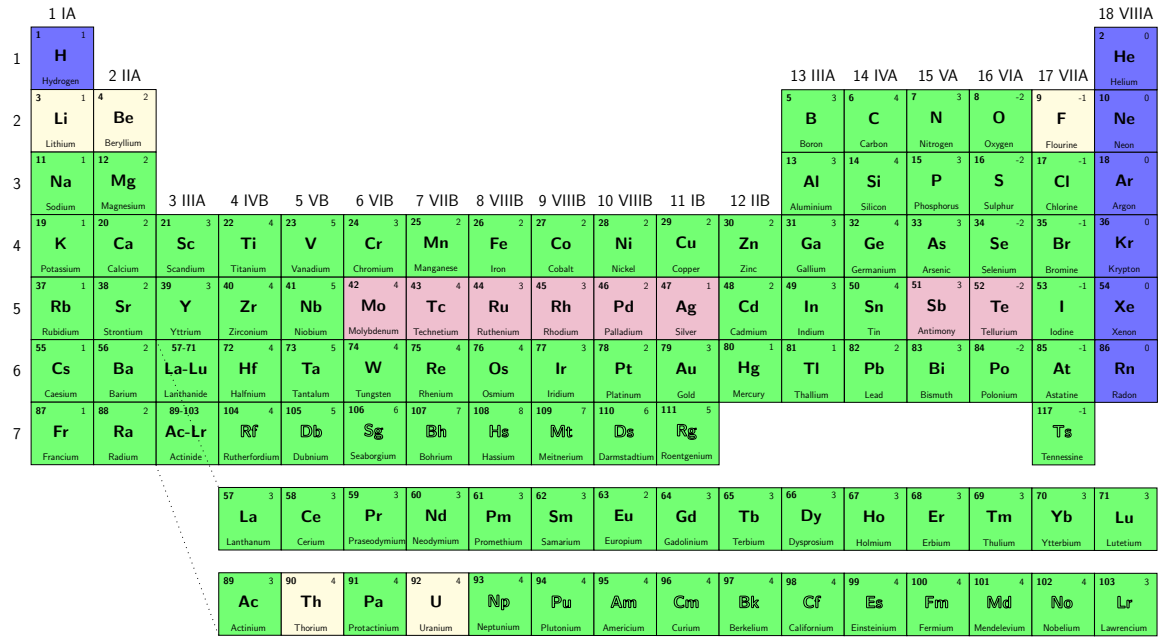


Figure 3.1: Elements in blue are those which bubble out of the salt and are included in the natural processes removal stream as are elements in light-red as these are the 'noble' metals which are insoluble in the salt. Elements in green are considered fission products of which 50% are removed every ten years. Periodic table layout in TikZ provided by Chris Rump and Ivan Griffin through *overleaf.com*.

learned from these figures and outputs are given. The units given for group and stream values, $\% - \rho_i^{mat}$, are relative fractions of the host material, mat, density as measured at the beginning of burnup step i.

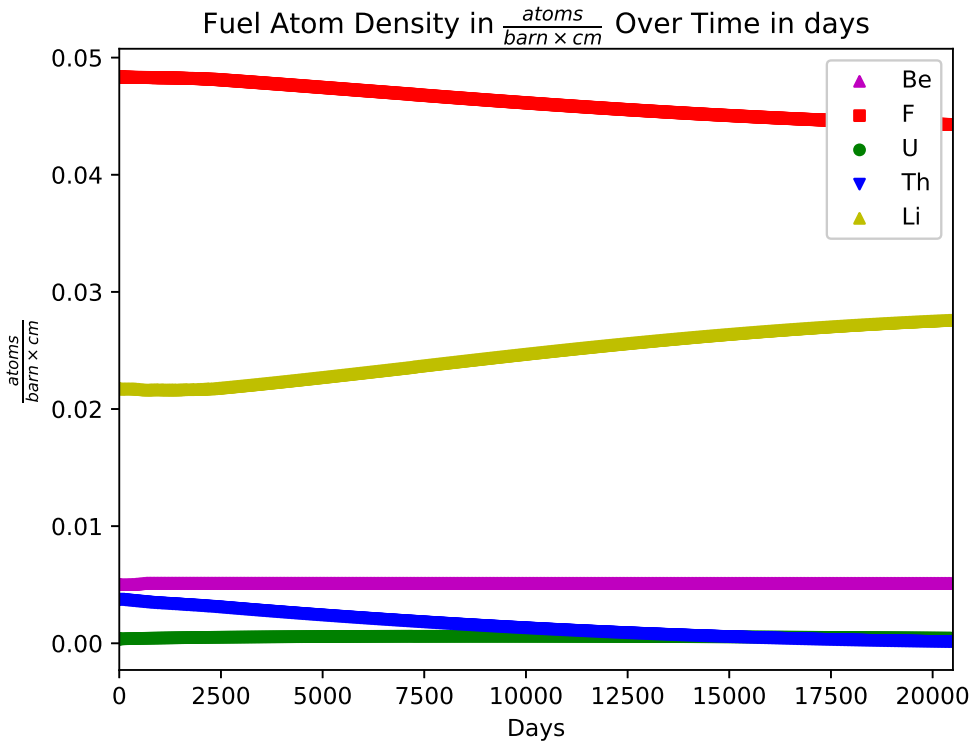


Figure 3.2: The primary fuel constituents - Li (yellow), Be (purple), F (red), U (green), Th (blue) - atom density over the entire simulation.

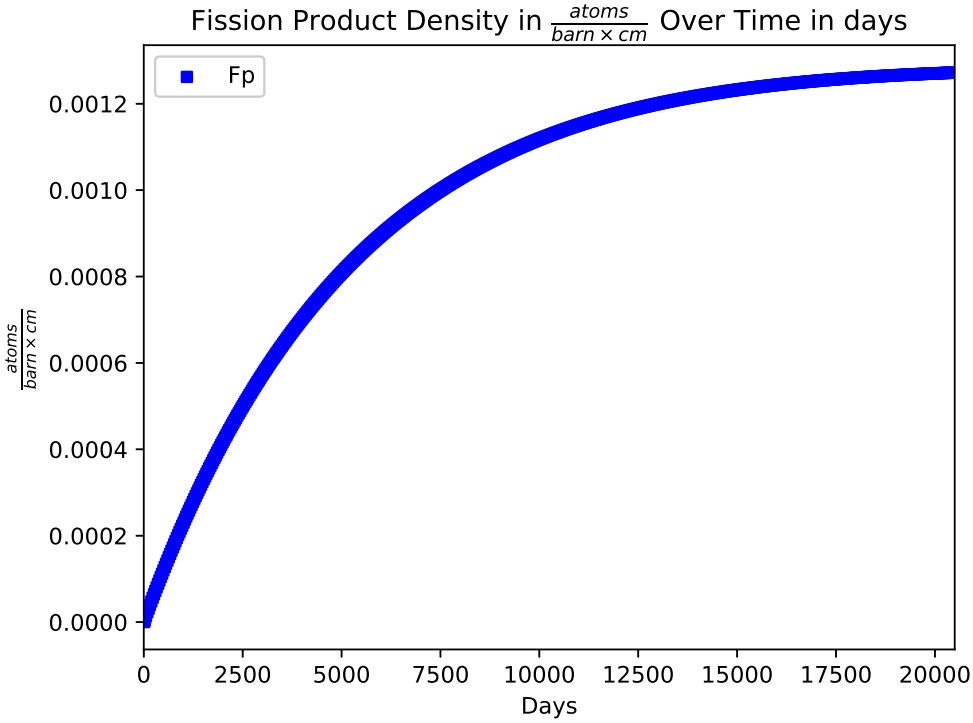


Figure 3.3: As described in section 3.1 and visually seen in figure 3.1 here is seen the atomic density of all elements in the salt considered to be fission products.

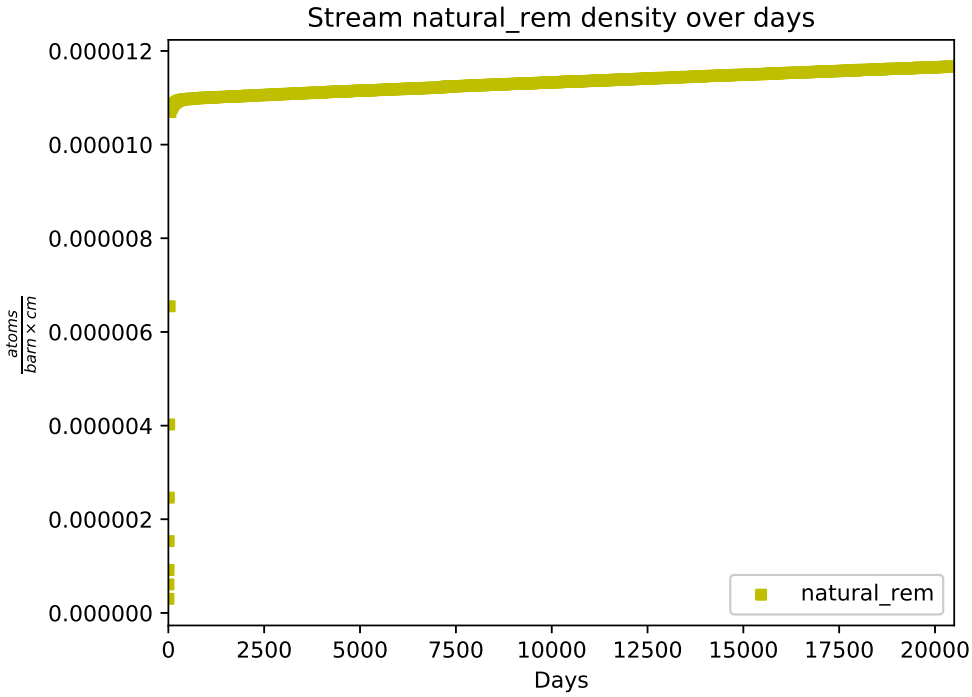


Figure 3.4: The atomic density of the sum total of elements removed from the system by the stream used to simulate natural removal processes.

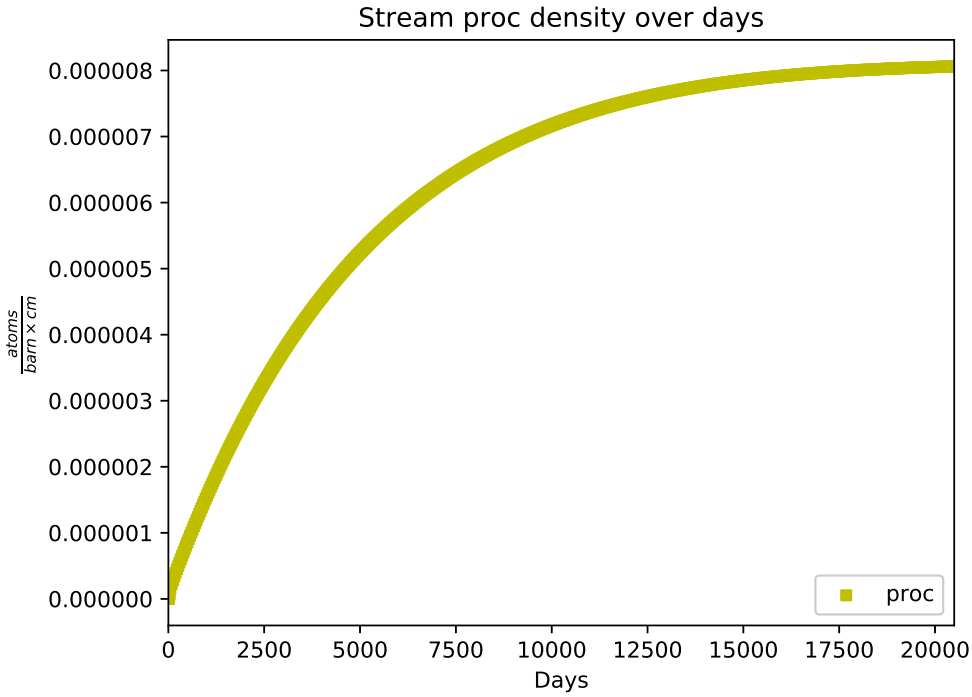


Figure 3.5: The atomic density of the sum total of elements removed from the system by the stream used to simulate a reprocessing function applied to the fuel salt.

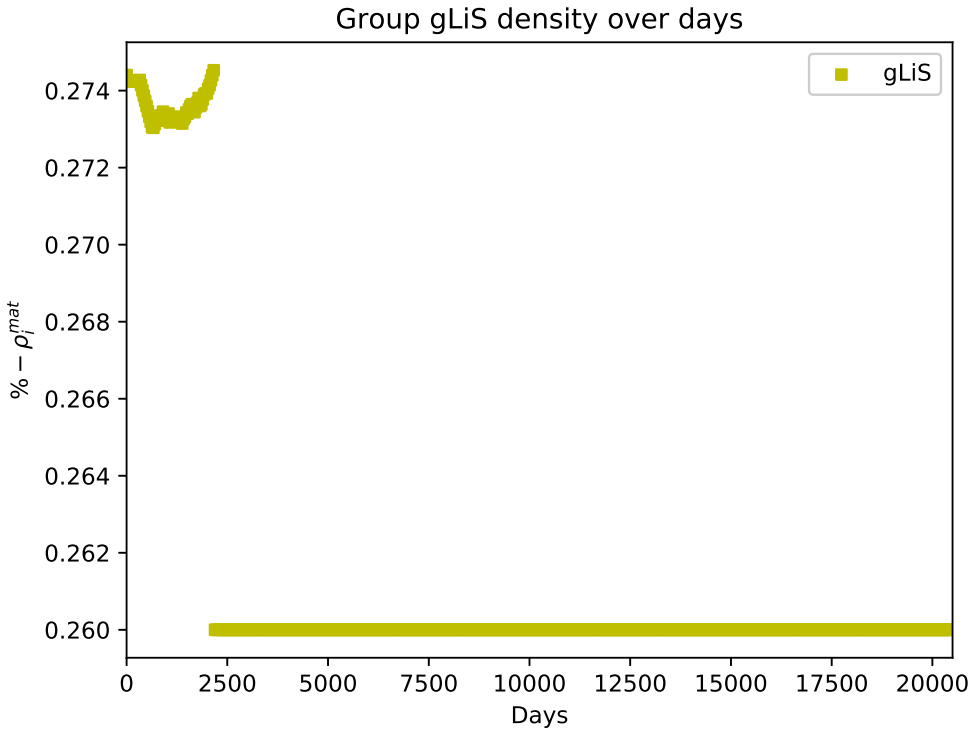


Figure 3.6: The atomic density of lithium elements considered to be a primary salt constituent.

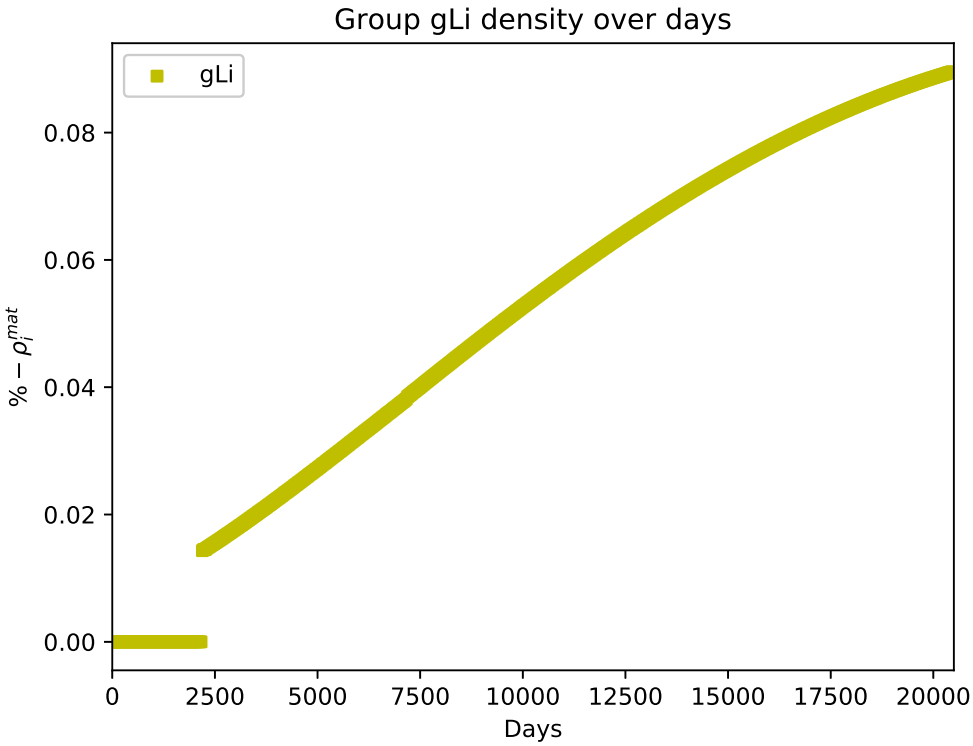


Figure 3.7: The atomic density of lithium elements excepting those considered to be a primary salt constituent.

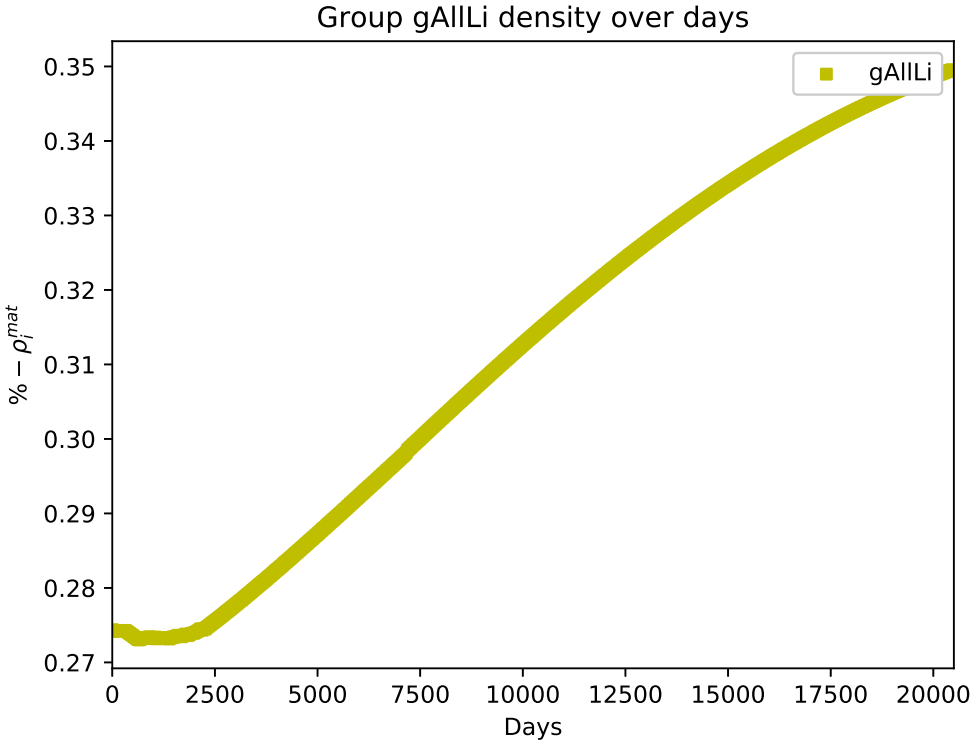


Figure 3.8: The atomic density of all lithium isotopes in the simulation.

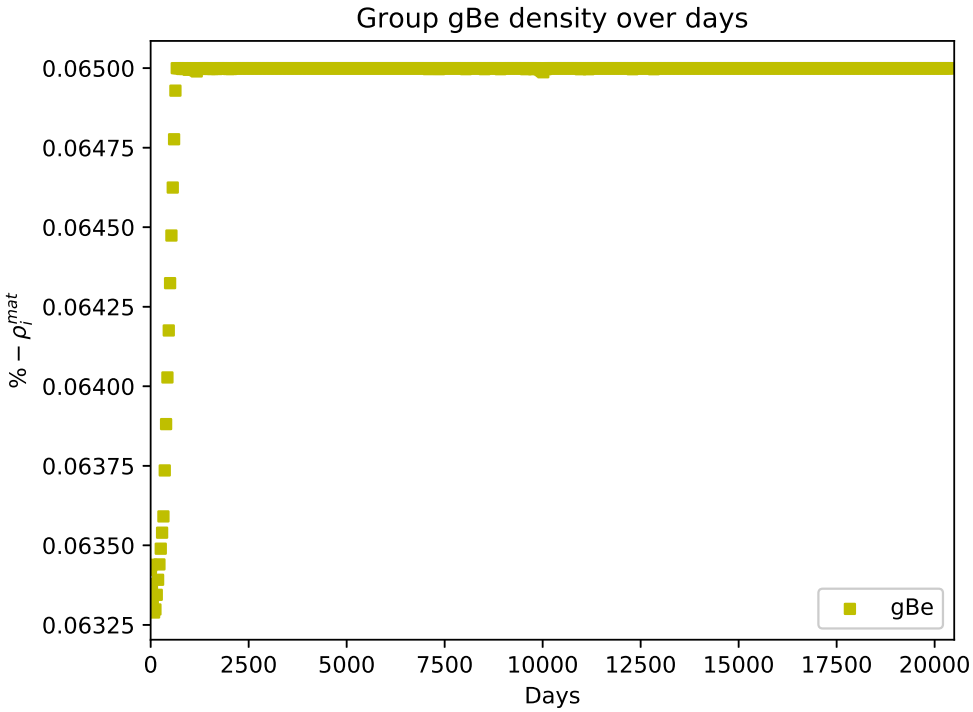


Figure 3.9: The atomic density of all beryllium isotopes in the simulation.

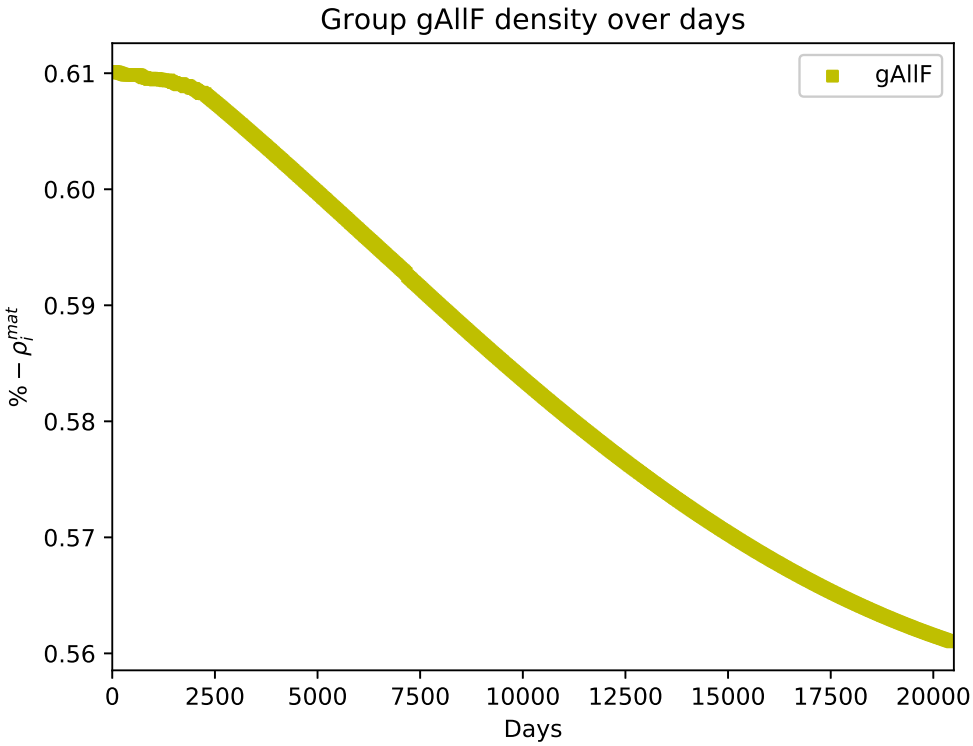


Figure 3.10: The atomic density of all fluorine isotopes in the simulation.

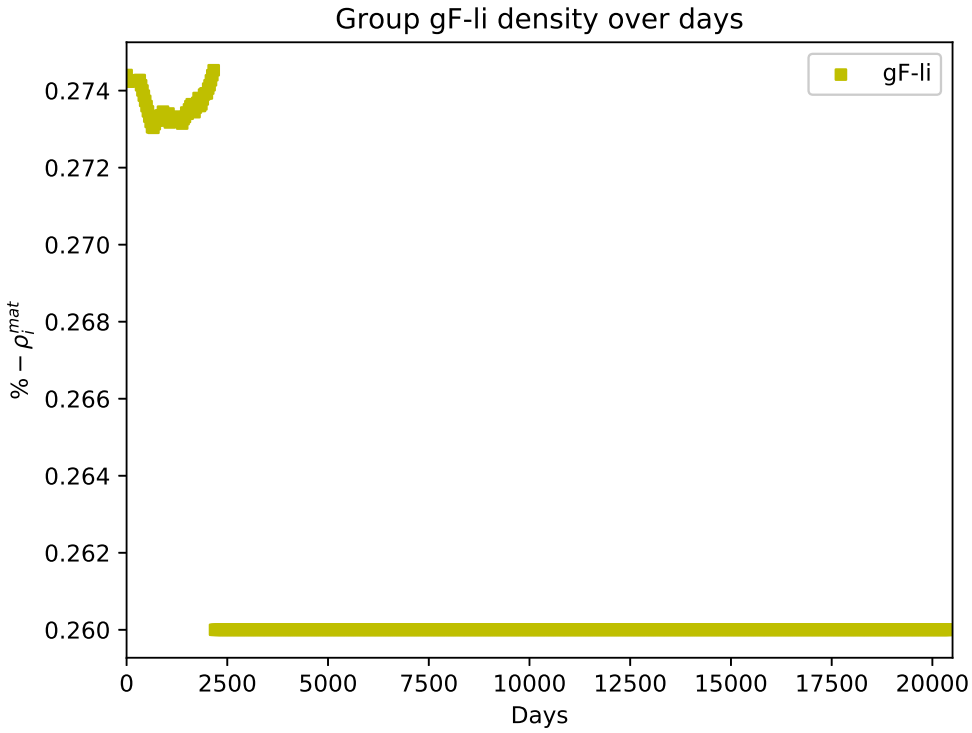


Figure 3.11: The atomic density of all fluorine isotopes considered to be bonded to lithium atoms which are considered to be primary fuel salt constituents.

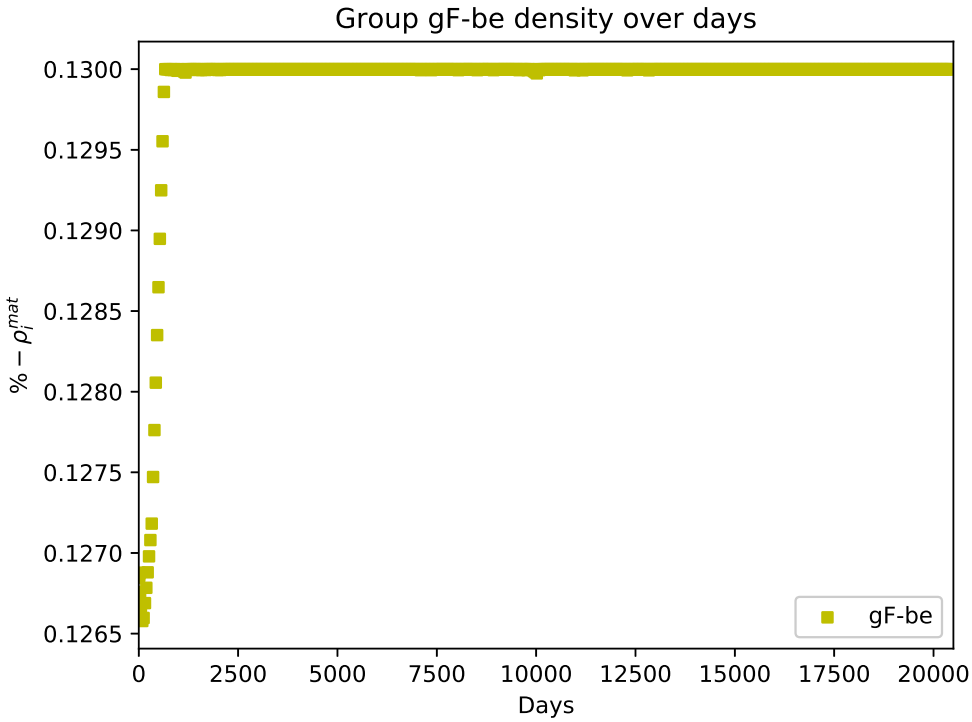


Figure 3.12: The atomic density of all fluorine isotopes considered to be bonded to beryllium atoms.

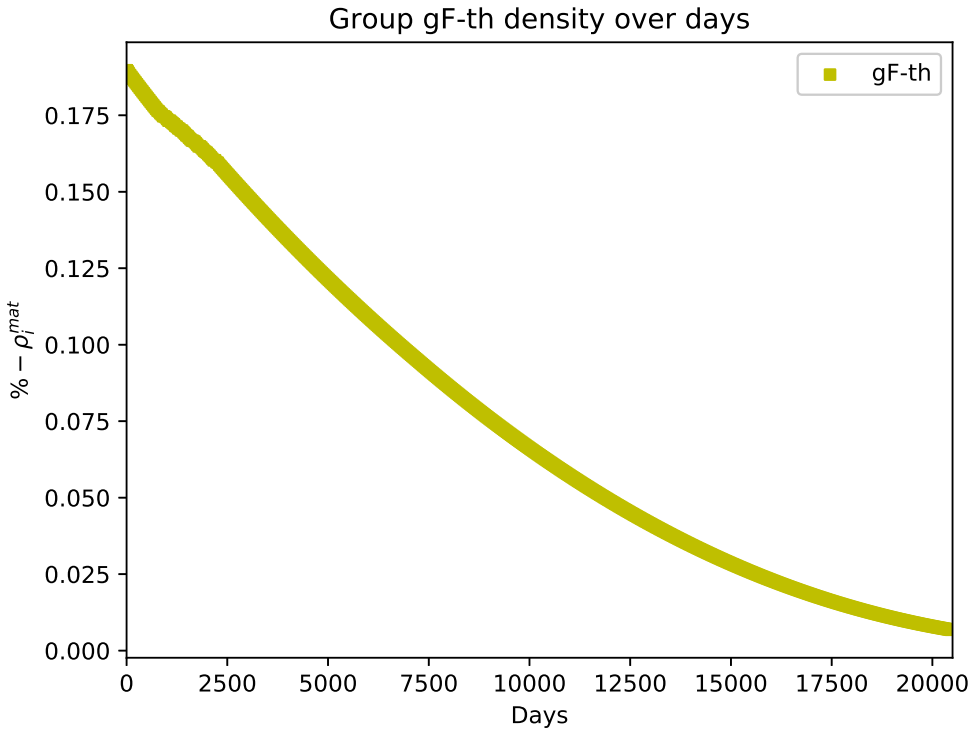


Figure 3.13: The atomic density of all fluorine isotopes considered to be bonded to thorium atoms

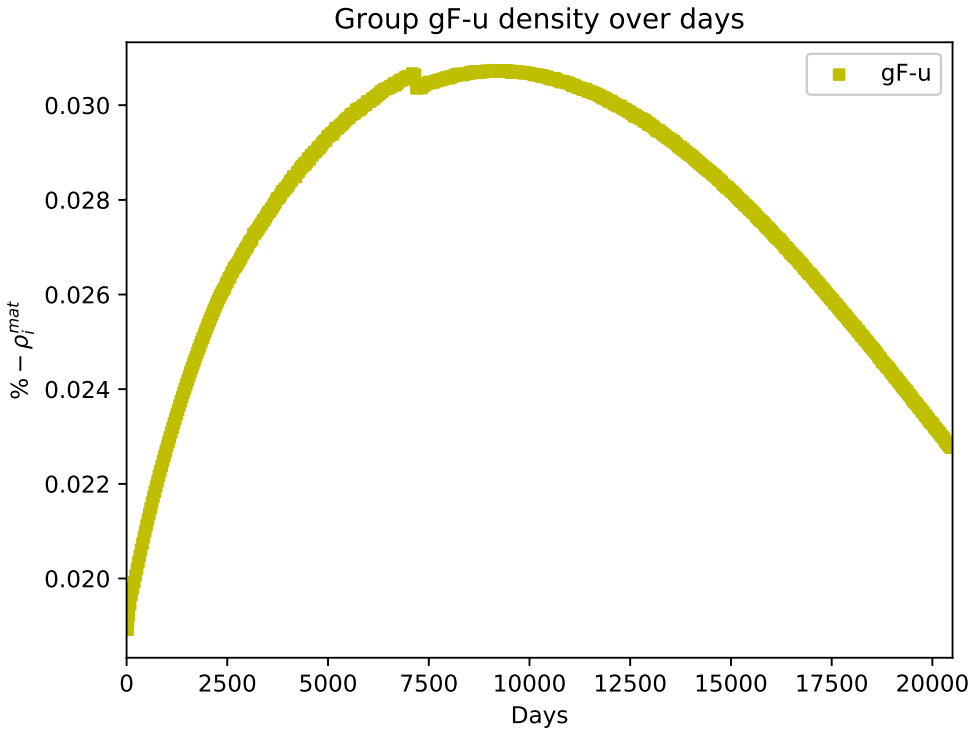


Figure 3.14: The atomic density of all fluorine isotopes considered to be bonded to uranium atoms

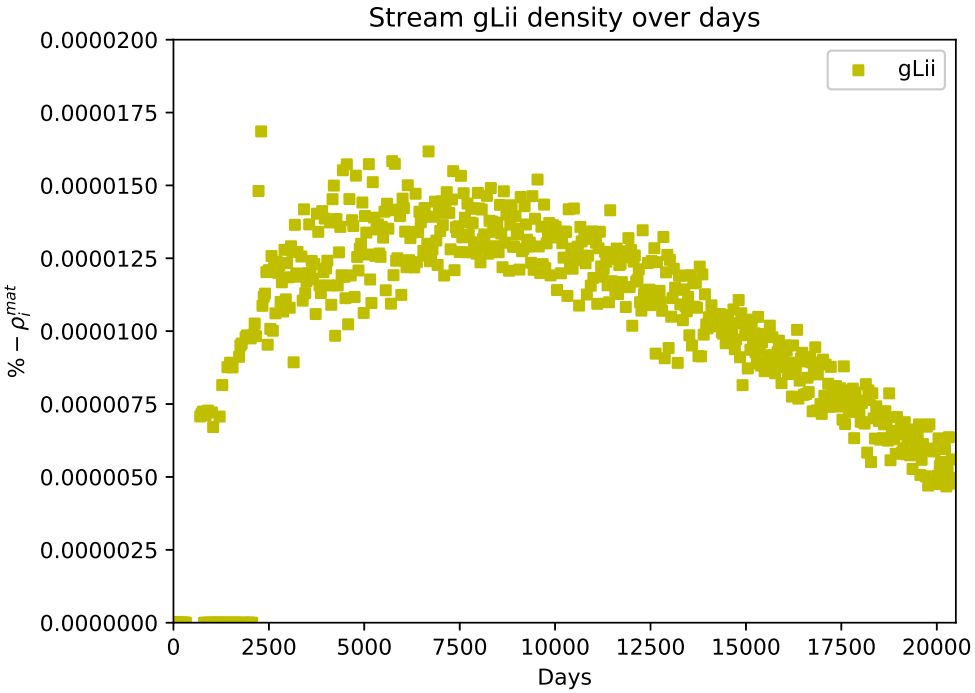


Figure 3.15: The deposition rate integrated over the time of burnup step i for the stream injecting lithium isotopes into the system.

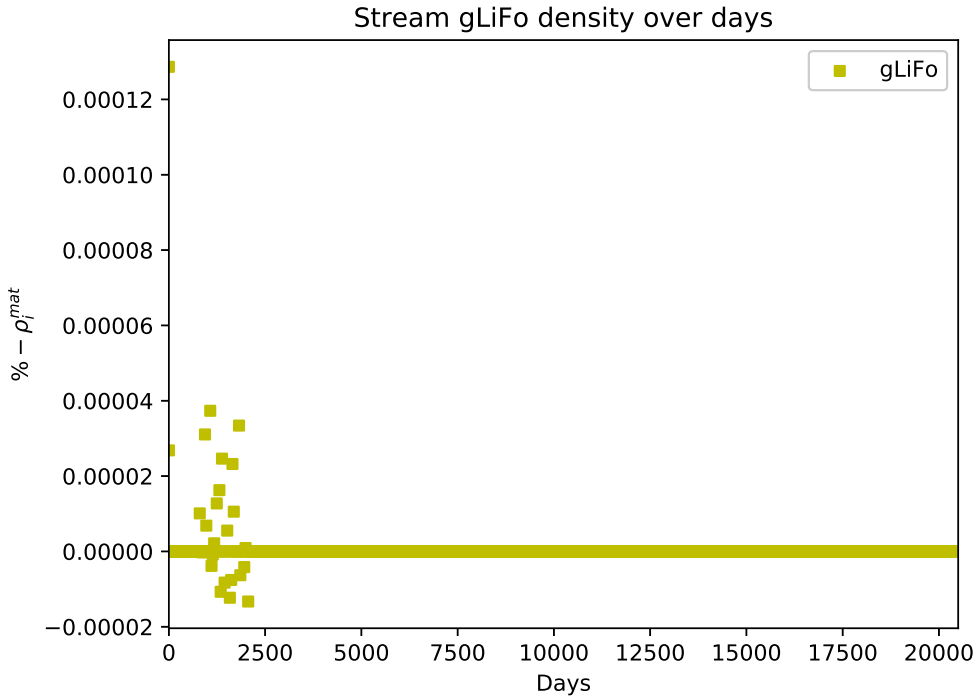


Figure 3.16: The removal rate integrated over the time of burnup step i for the stream removing LiF from the system.

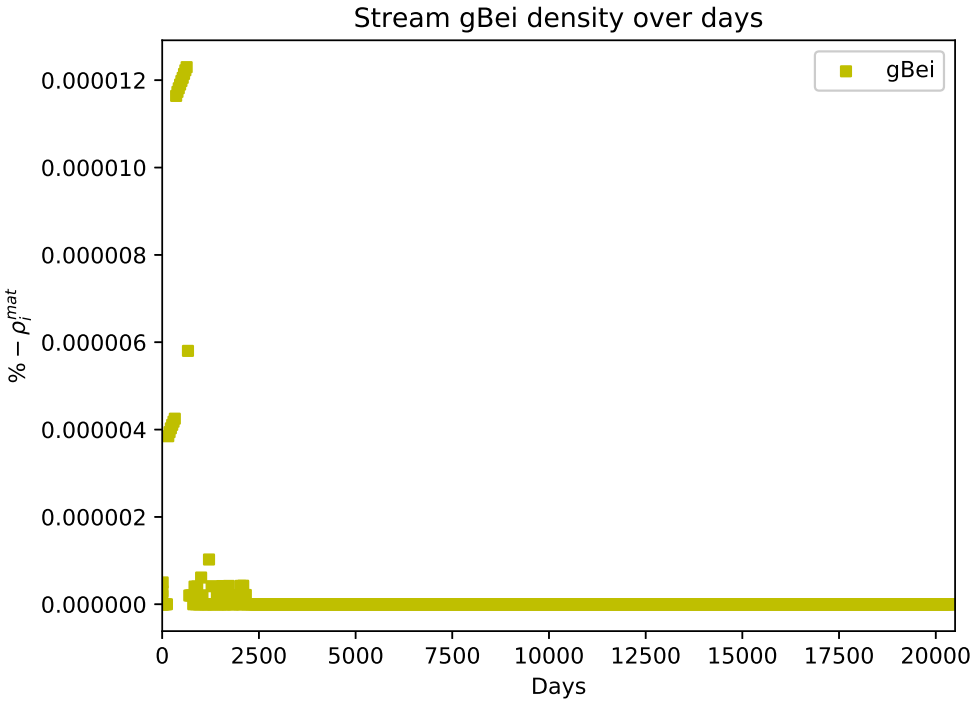


Figure 3.17: The deposition rate integrated over the time of burnup step i for the stream injecting beryllium-9 into the system.

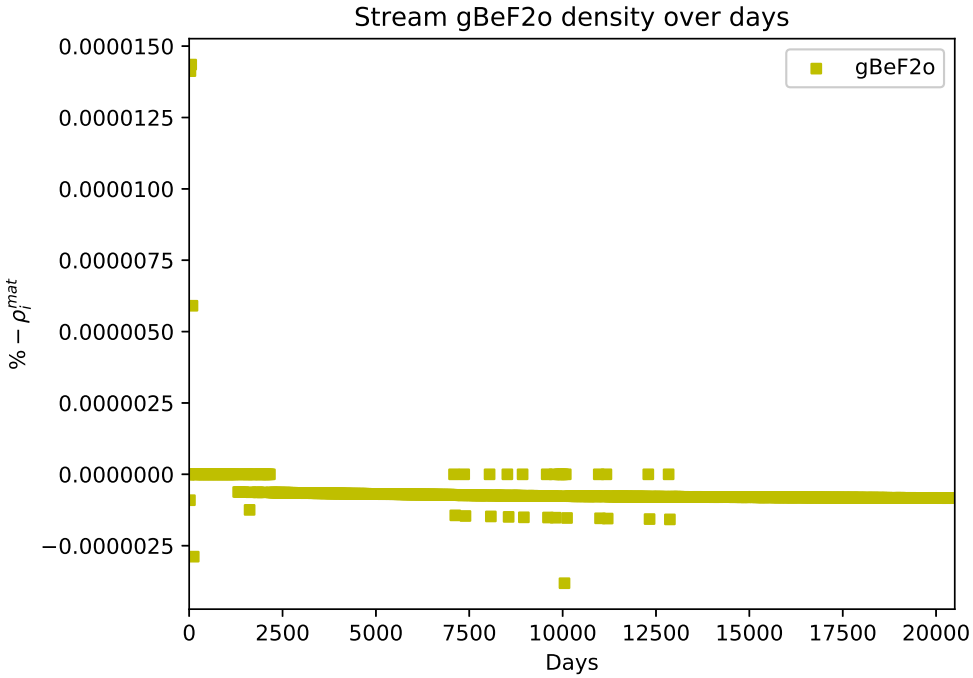


Figure 3.18: The removal rate integrated over the time of burnup step i for the stream removing BeF_2 from the system.

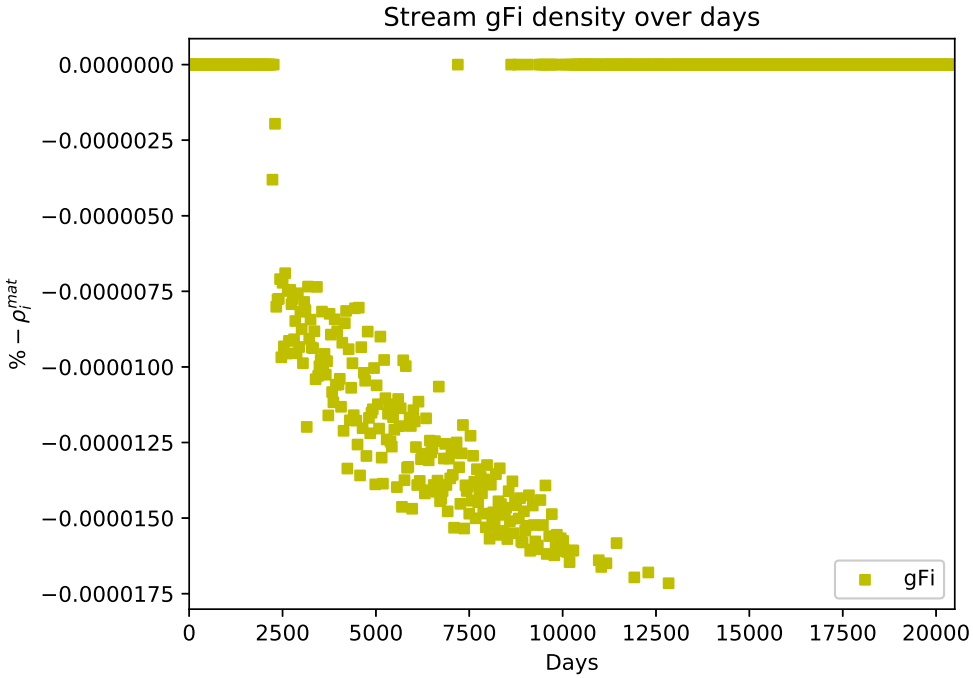


Figure 3.19: The deposition rate integrated over the time of burnup step i for the stream injecting fluorine-19 into the system.

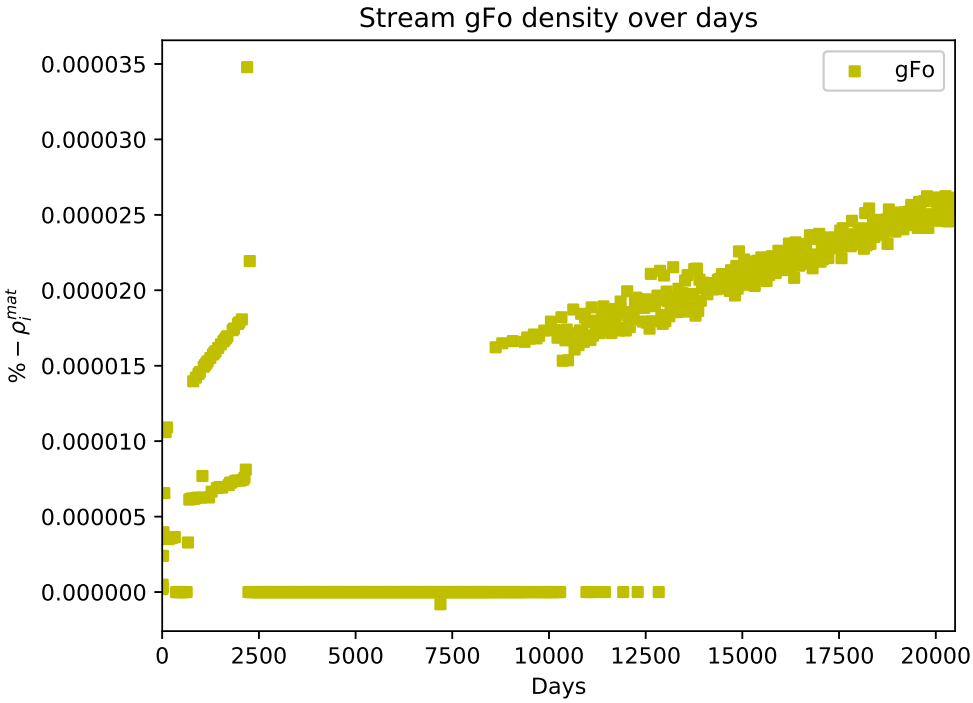


Figure 3.20: The removal rate integrated over the time of burnup step i for the stream removing fluorine from the system.

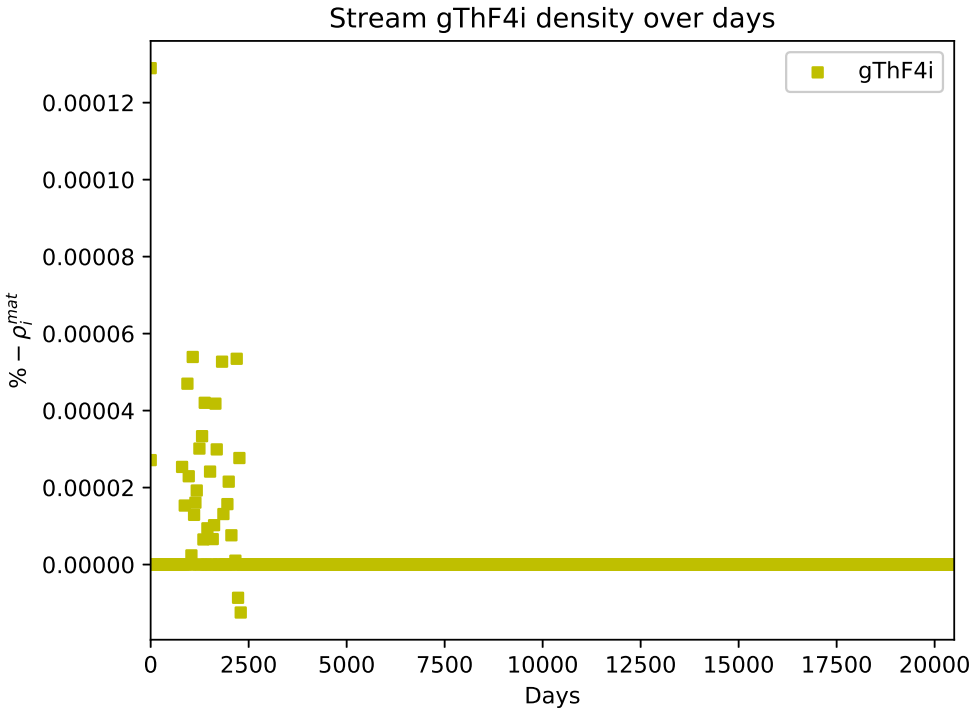


Figure 3.21: The deposition rate integrated over the time of burnup step i for the stream injecting $^{232}\mathrm{Th}^{19}\mathrm{F}_4$ into the system.

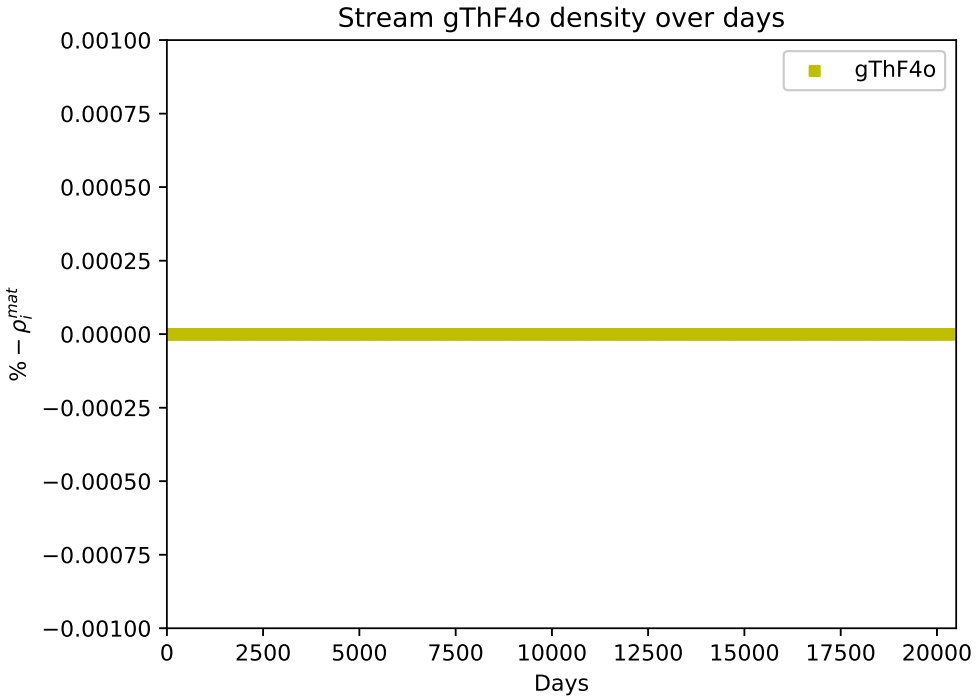


Figure 3.22: The removal rate integrated over the time of burnup step i for the stream removing ThF_4 from the system.

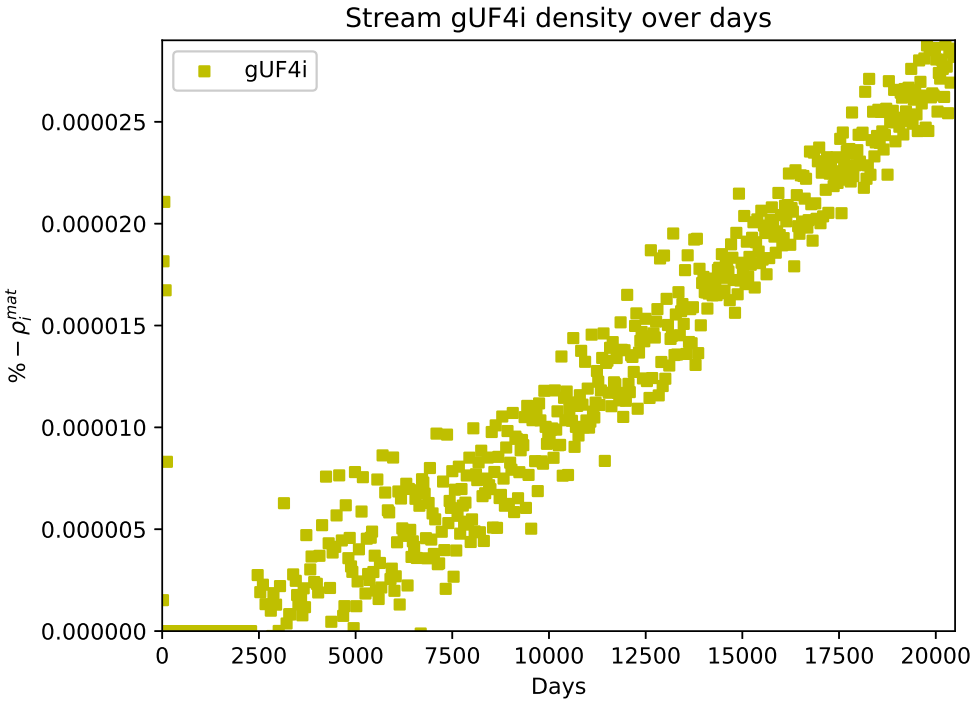


Figure 3.23: The deposition rate integrated over the time of burnup step i for the stream injecting $^{233}U^{19}F_4$ into the system.

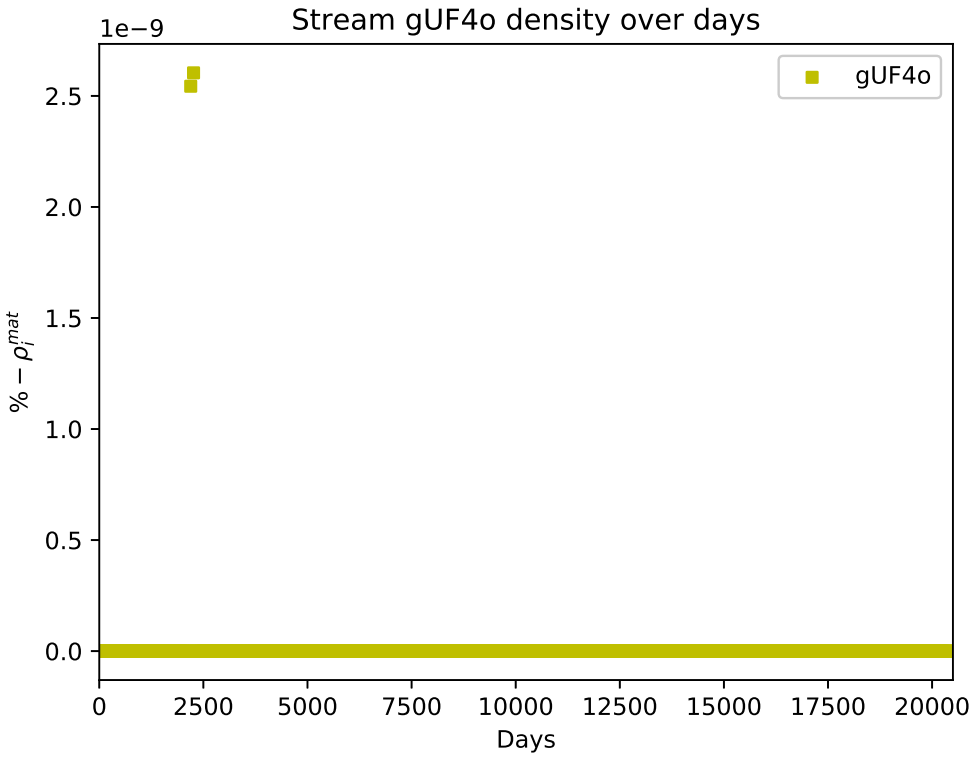


Figure 3.24: The removal rate integrated over the time of burnup step i for the stream removing UF_4 from the system.

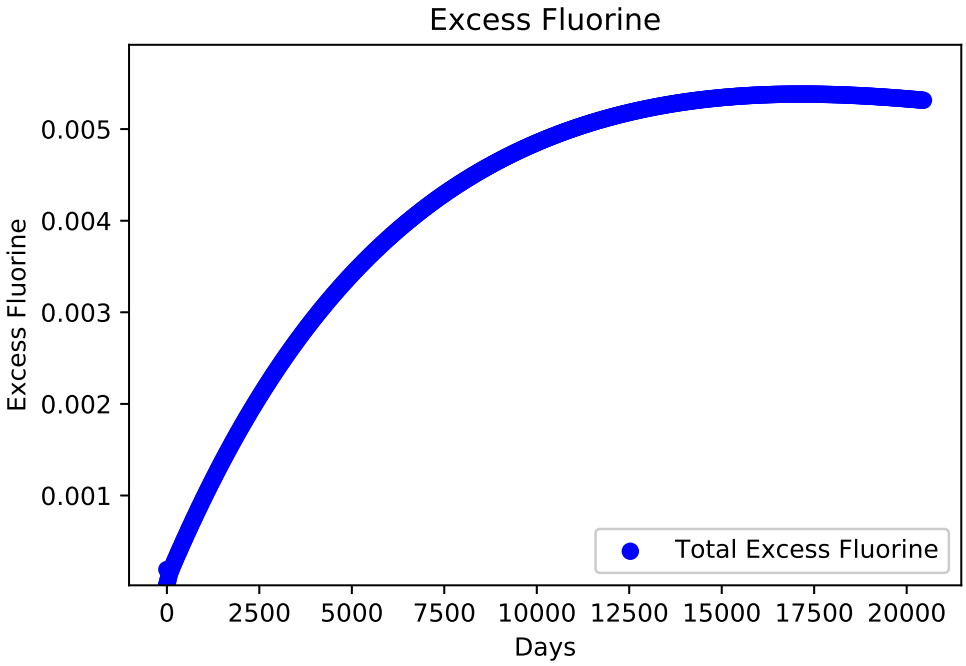


Figure 3.25: The amount of fluorine over time in excess of the amount of fluorine required to bind to all the primary salt constituents: LiF, BeF_2, ThF_4, UF_4.

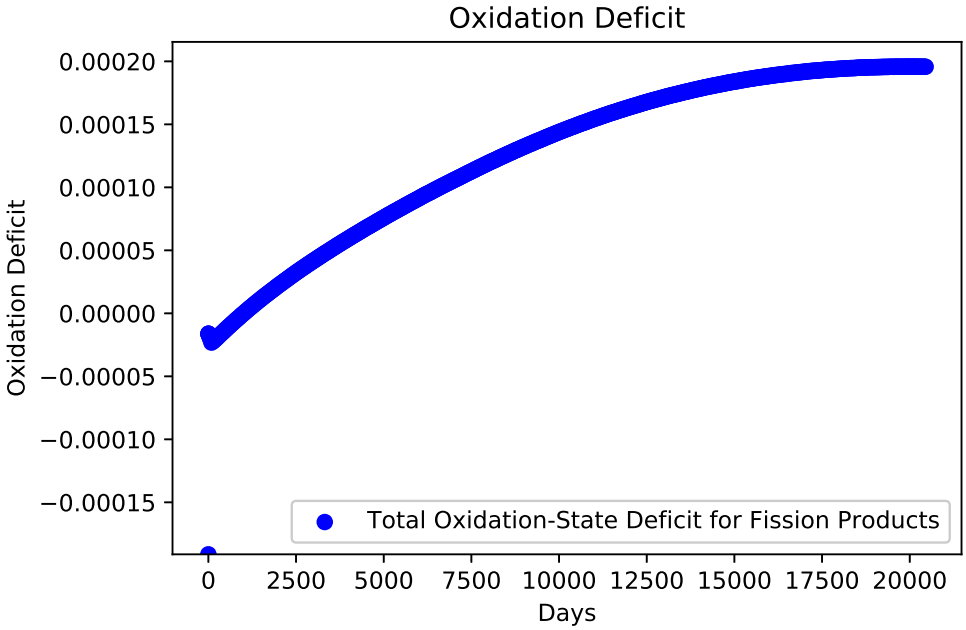


Figure 3.26: The simulation's summed oxidation state over all elements and weighted by the prevalence of that elements, as in equation 2.10, over time.

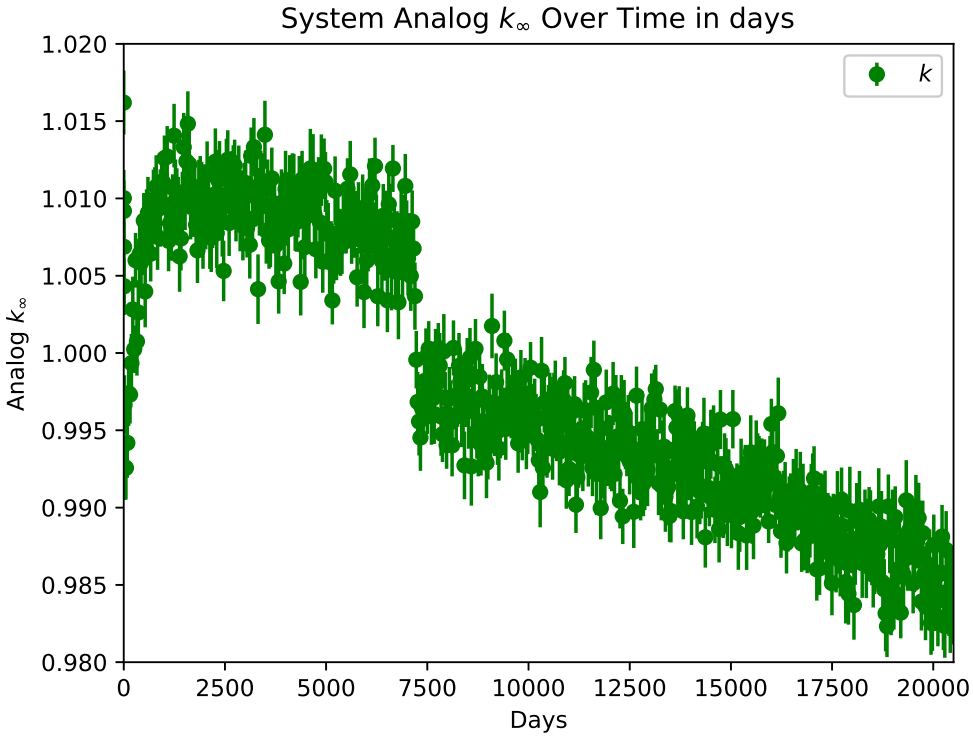


Figure 3.27: The observed infinite neutron multiplication factor of the system complete with error. The very first time point at $t = 0$ is excluded from this plot for scale reasons. The initial infinite neutron multiplication factor of the system was approximately 1.5.

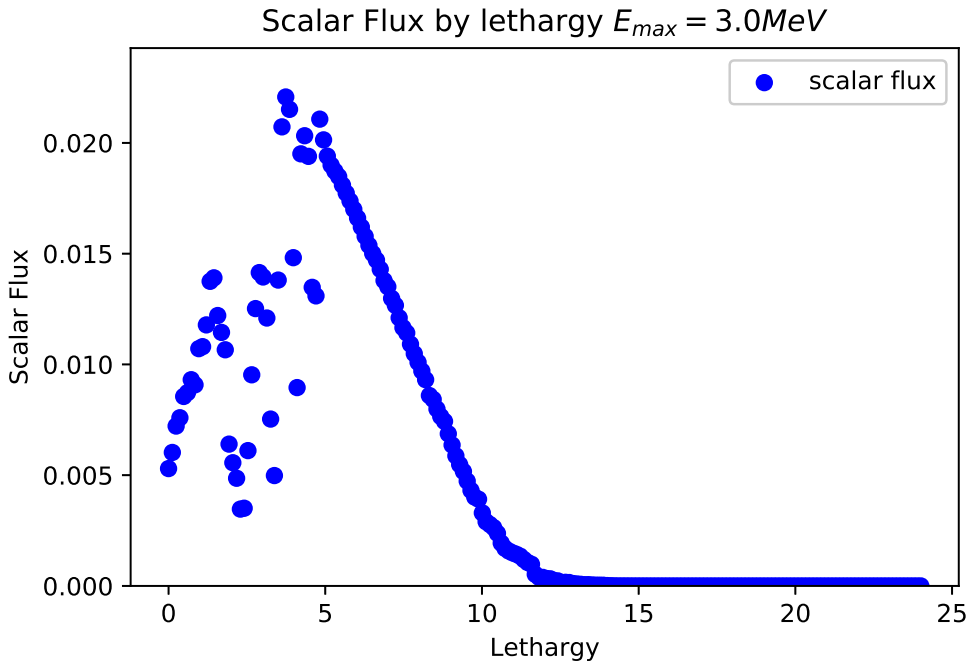


Figure 3.28: The scalar neutron flux within the system broken up into equal-lethargy width bins taken at day zero.

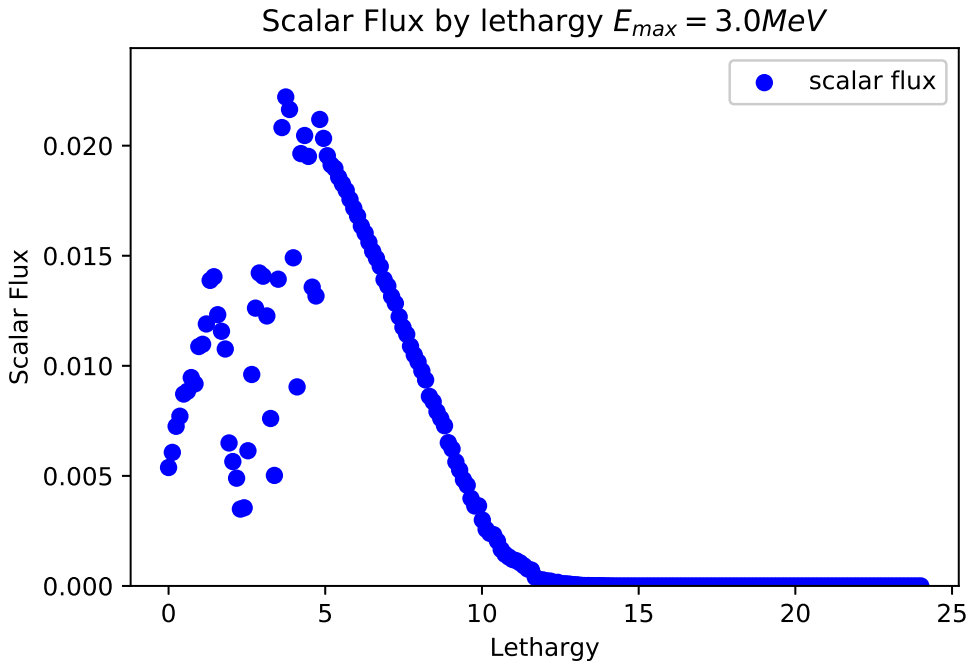


Figure 3.29: The scalar neutron flux within the system broken up into equal-lethargy width bins taken at burnup step 219 - the center of the discontinuity.

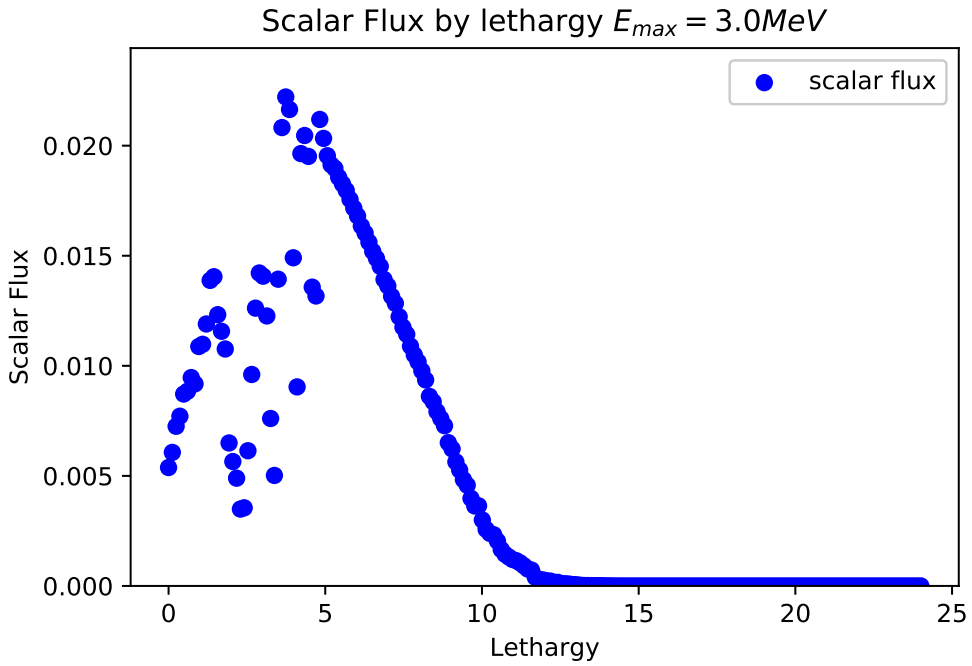


Figure 3.30: The scalar neutron flux within the system broken up into equal-lethargy width bins taken after the last burnup step.

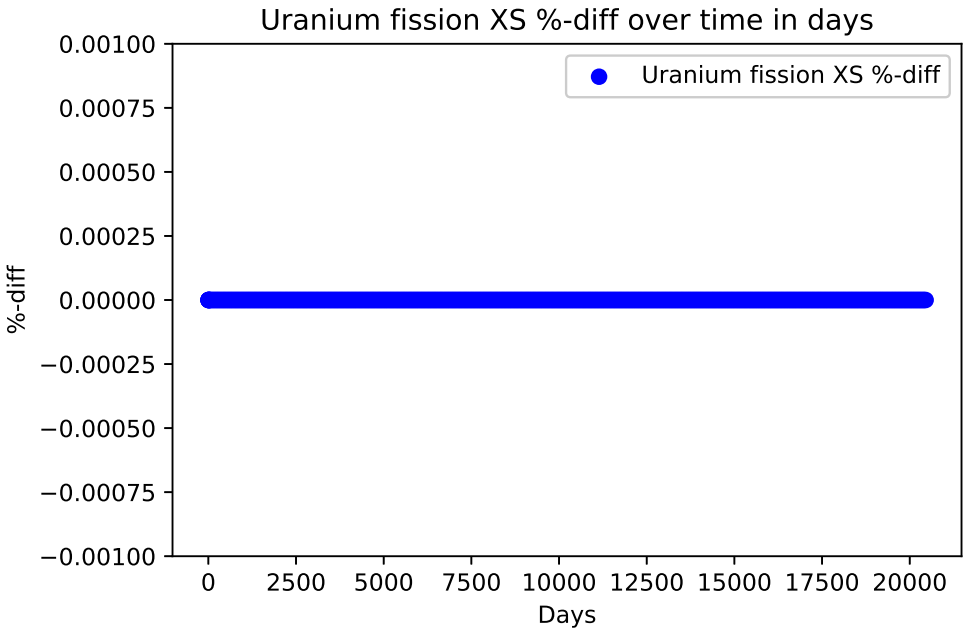


Figure 3.31: The relative percentage change in the fission cross section of ^{233}U per burnup step throughout the simulation.

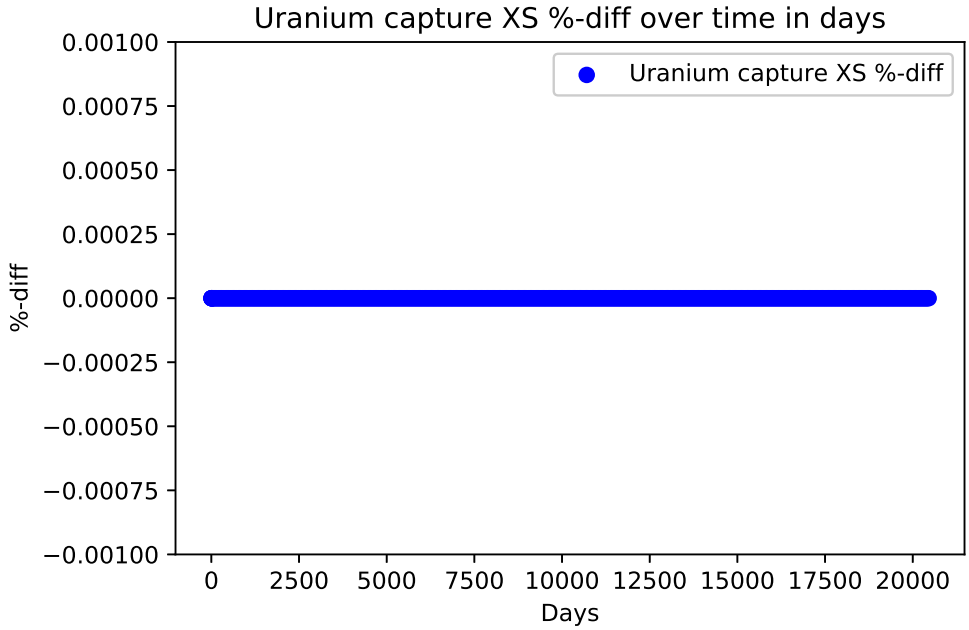


Figure 3.32: The relative percentage change in the absorption cross section of ^{233}U per burnup step throughout the simulation.

3.3 Simulation Assessment

In figure 3.2 the elemental salt constituent atomic densities for Li, Be, F, Th, and U are plotted over time. This plot represents a turning point in MSR modelling, one after which incorporating chemistry restraints into nuclear burnup calculations is possible. Beginning with fluorine as the most abundant salt constituent a steady decline is observed - at first very slight but growing in magnitude towards the end of the simulation. At an exaggerated scale figure 3.10 magnifies this trend.

Throughout this section a theme that will emerge are the apparent shortcomings of using the CLP linear optimization library for such an endeavor. While Sandia national labs tested CLP themselves, [12], and my own stress tests as well as ADER's unit and integration tests confirmed that CLP could and would handle the floating point precision involved in these kinds of problems, it was not until years later that Ma and Saunders would show in [13] that the numerical instability issues do not arise until many thousands of parameters are involved in the calculations. The input for this simulation was checked in a debugger at every stage of program execution and was found to be consistent with the expected behavior given the input seen in appendix C. Indeed, the linear optimization matrix as requested by CLP was checked and found to be consistent with describing the input to CLP. Despite this the behavior from CLP, as observed in the previous figures, indicates a solution which ignores parameters as well as crashing all together on a whole host of other problems.

A prime example of this behavior is seen in figure 3.19 where the injection stream for fluoride-19 is seen to take only negative values other than zero. A hard lower bound of zero was applied to all streams in the CLP solution. An interesting observation is that this 'negative injection' by this stream would seem to fill in the missing trend seen in figure 3.20 and indeed accounts for some of the change observed in figure 3.10. Figure 3.20 does have the expected behavior in reference to figure 3.10. While CLP ignores some parameters it does follow others as exemplified in figure 3.25 where it can be seen that throughout the entire simulation sufficient fluorine was kept in the system to bind to all primary fuel constituents where excess fluorine is taken to be the fluorine unaccounted for in the system after summing all fluorine required to bond to the primary fuel constituents. Figures 3.11, 3.12, 3.13, and 3.14 confirm that the respective fluorine groups track their bonded element at the appropriate ratios.

The behavior seen for beryllium follows a similar trend with the removal stream for BeF_2 having negative values corresponding to a brief positive trend in the injection stream for beryllium as can be seen in figures 3.18 and 3.17 respectively. This behavior is consistent with the maximization for the beryllium fuel group by ADER in the initial days of fission product production to counter the produced negative reactivity by further thermalizing the spectrum with beryllium.

This observation is further backed by the initial loss of lithium in the early steps of the simulation as seen in figures 3.6 and 3.8 - again to increase system reactivity as lithium-6 is a strong neutron poison. Again slight negative values are seen in a stream as can be observed in figure 3.16. The increased injections of lithium into the system, accounting for the rise seen in figure 3.10, are seen in figure 3.15. From figure 3.7 it can be seen that this increase in lithium was not for primary salt constituent balance purposes. Rather, this increase in lithium, along with the expected increase in fission products which are entirely oxidizing to the salt, accounts for the overall system increase of the weighted oxidation state value as seen in figure 3.26.

This continued increase of the weighted system oxidation state up to the positive asymptote observed at 0.004 a.u. is a complete violation of the hard oxidation state value limits that were set for this simulation of $[-0.0002, -0.0001]$. Furthermore it would seem that this system oxidation state was actually pursued by CLP as there is no other reason to be injecting additional lithium, a neutron poison, into the system.

The initial injections of thorium into the system as seen in figure 3.21 are produced to account for the unacceptably high initial system neutron multiplication factor as can be seen in figure 3.27 - even though in that figure the very first point at $[k = 1.06, t = 0]$ has been cut off for scale readability purposes. Interestingly enough the thorium removal stream maintains a perfect zero value throughout the simulation indicating that the thorium depletion seen in figure 3.2 is due entirely to nuclear processes to the degree that would be expected.

Looking to uranium the two errant values of positive uranium removal seen in figure 3.24 have no explanation other than possible CLP numerical instability. On the other hand, the steady increase in uranium injection as seen in figure 3.23 is entirely expected given the

observed increase in fission products, a powerful neutron poison, in figure 3.3.

The behavior of the natural processes removal stream, as seen in figure 3.4 follows the expected pattern with a rapid rise to its saturation value as the equilibrium content of short lived - or rapidly removed - fission products is approached. The behavior of the reprocessing stream as seen in figure 3.5 agrees with the behavior observed for the overall fission product concentration as seen in figure 3.3 and for which the two of them explain the fission product concentration approach to its equilibrium value. This behavior is expected as table-class streams have their transfer values calculated by ADER routines, not by CLP. This information is then passed into the CLP simulation but does not affect the behavior of said table-class streams.

In the first 6000 days of the simulation ADER maintains the neutron multiplication factor within the requested bounds as seen in figure 3.27 through the application of streams as seen in the figures previously discussed. However, over a series of burnup steps, 216 - 221, the neutron multiplication factor of the system falls quickly, and then while remaining below the desired neutron multiplication factor targets, continues in a downward trend despite a continuing increase in the amount of ^{233}U injected into the system as seen in figure 3.23. As seen in figures 3.31 and 3.32 the fission and capture cross sections of ^{233}U do not change through the entire simulation. Furthermore, as can be seen in figures 3.28, 3.29, and 3.30 the neutron spectrum does not change through the entire simulation and not in the middle of the discontinuity either. In debugging efforts no deviation from previous burnup steps or later burnup steps in the evaluation of nuclear cross sections by ADER was found however ADER consistently, from the discontinuity onwards, chose material configurations that by its own calculations would drive the system neutron multiplication factor below the desired minimum target of 1.00.

Overall these results represent a mixed bag. Obviously ADER's use of CLP suffers from pernicious and extensive numerical instability issues. These issues manifest not just in aborted simulations but in simulation results which do not adhere to the system parameters as defined. Despite these shortcomings ADER does execute some of its intended functions such as general chemical solubility control and species accounting. It performs exceptionally well with regards to the behavior and incorporation of table-class streams. It is worth noting here that a simulation with no group-class streams will bypass the CLP engine all together and may result in a numerically stable simulation - one which would would have no features which can not already be found in popular nuclear depletion codes such as ORIGEN.

Nonetheless ADER demonstrates the potential application and effectiveness of such an algorithm for a more comprehensive approach to MSR fuel cycle analysis.

Chapter 4

Conclusions and Future Work

In the last twenty years molten salt reactors have seen an incredible surge of global attention as evidenced in such international programs as MOSART, ALISIA, and the LF-TMSR out of China, as well as increased research attention as evidenced by the continuing success of Oak Ridge National Laboratory's MSR workshop which now averages over 400 attendees from across the globe yearly. While significant investment and progress has been made in nearly all aspects of MSR development from material selection, salt purification and property measurement, licensing and general reactor design, comparatively little attention has been paid to fuel cycle analysis.

The majority of attempts in this area have fallen short of producing widely applicable results largely due to the limiting effects of specific assumptions such as the salt species involved or the coarseness of the solution. In this work a method and implementation of a general approach for modelling and evaluating molten salt reactor fuel cycles is proposed. This approach is based upon a linear optimization routine designed to approximate the limitations of the chemistry and nuclear concerns of reactor operations while optimizing the driving concerns of the reactor operators. This approach, named ADER for the Advanced Depletion Extension for Reprocessing, is built into the reactor physics Monte-Carlo code SERPENT 2.

ADER brings to the users of SERPENT 2 the ability to model several aspects of MSR operations and fuel cycle analysis through the introduction of several vital utilities. The first among these is the ability to define collections of elements, isotopes and chemicals as well as the ability to set absolute and relative abundance constraints between any set of these collections, called groups. In order that ADER might push material compositions towards those which satisfy the group constraints ADER provides to the user the ability to define mass transfers into, out of, and between materials in a SERPENT 2 simulation. To ensure that these mass transfers do not disturb the desired neutron multiplication factor in the system ADER allows the user to set neutron multiplication maximum and minimum values. During the material optimization phase ADER estimates the reactivity impact of its selected mass transfers and adjusts these transfers to keep the system within the desired bounds. Furthermore ADER provides the ability for the user to set the desired bounds

of the averaged oxidation state of materials in a simulation such that ADER will keep its selected mass transfers from disturbing the redox potential of the material, a critical value in determining the effects and extent of corrosive activity. Bringing all of this together is an easy to use and directly integrated user interface supported by a full suite of tests as well as documentation.

In chapter 3 the results of a simple simulation employing a $LiF-BeF_2-ThF_4-UF_4$ salt mixture in an infinite and homogeneous medium are presented. While this simulation did not cause CLP to crash and error out, nonetheless the results obtained from the simulation present a mixed view of ADER's implementation. ADER is certainly seen to influence the outcome of the depletion simulation, adding uranium salts to maintain the minimum value of the neutron multiplication factor in the system. However, occurring around burnup step 219, ADER's corrections fail to bring the neutron multiplication value of the system up to at least the minimum value. ADER did maintain the necessary amount of fluorine in the system to bind to all primary salt constituents. However, the mass flows ADER employed to accomplish this took on non-physical values. With regards to the corrosion monitoring feature of ADER, these limitations were completely ignored by the simulation for an unknown reason likely having to do with the inherent numerical instability within the optimization library. Despite these shortcomings this simulation has undoubtedly shown the value of an algorithm for incorporating the concerns which ADER addresses into a nuclear fuel depletion simulation. As put forth in chapter 2 the implementation of a quad-precision linear optimization solver should, according to [13], resolve the numerical instabilities plaguing ADER - at which point the algorithm is expected to show great utility in molten salt reactor analysis.

Any future work going forth on this project would need to begin with the implementation of said floating-point quadruple-precision linear optimization solver in the place of the current CLP implementation. Following such a development it is expected that ADER will be capable of simulating the wide variety of parameters which its input and structure allow. Additional improvements could be found through the incorporation of an iteration scheme whereupon the effects of nuclear burnup on the isotopics of the fuel over a burnup step can be approximated as a proportional removal stream on the whole system in the linear optimization scheme such that the approximated total effects of nuclear burnup are considered in material optimization.

Overall the impact of ADER is clear in that it represents a more complete approach to MSR fuel cycle modelling. Given ADER's extensive test suite, documentation, and modular construction, it is not unimaginable that the above mentioned improvements may one day be made. In such a future the impact of ADER would certainly be greater. Despite this shortcoming ADER has shown that a linear optimization scheme can be effectively applied to the chemistry, nuclear, and operational concerns of a molten salt reactor in such a way as to predict the future fuel composition and nuclear characteristics of the system.

Bibliography

[1] M. Rosenthal, P. Haubenreich, R. Briggs, The development status of molten-salt breeder reactors, Oak Ridge National Laboratory Report 4812.

[2] V. Ignatiev, Development status of the gif msr system, International Atomic Energy Agency Report 49 (45).

[3] J. Koger, Fluoride salt corrosion and mass transfer in high temperature dynamic systems, Corrosion.

[4] M. Aufiero, A. Cammi, C. Fiorina, J. Leppnen, L. Luzzi, M. Ricotti, An extended version of the SERPENT-2 code to investigate fuel burn-up and core material evolution of the Molten Salt Fast Reactor, Journal of Nuclear Materials 441 (1-3) (2013) 473–486. doi:10.1016/j.jnucmat.2013.06.026.

[5] G. Ridley, O. Chvala, A method for predicting fuel maintenance in once-through MSRs, Annals of Nuclear Energy 110 (2017) 265–281. doi:10.1016/j.anucene.2017.06.043.
URL `http://linkinghub.elsevier.com/retrieve/pii/S030645491730021X`

[6] B. Betlzer, J. Powers, A. Worrall, Molten salt reactor neutronics and fuel cycle modeling and simulation with scale, Annals of Nuclear Energy 101 (2017) 489–503.

[7] B. Rearden, M. Jessee, Scale code system, Oak Ridge National Laboratory Internal Report 6 (2005).

[8] M. DeHart, I. Gauld, M. Williams, High-fidelity lattice physics capabilities of the scale code system using triton, Joint International Topical Meeting on Mathematics & Computation in Nuclear Applications.

[9] M. Bell, Origen - the ornl isotope generation and depletion code, Oak Ridge National Laboratory Internal Report (4628).

[10] J. Leppnen, SERPENT 2.
URL `http://montecarlo.vtt.fi`

[11] R. Lougee-Heimer, The Common Optimization INterface for Operations Research, IBM Journal of Research and Development 47 (1) (2003) 57–66.

[12] J. L. Gearhart, K. L. Adair, R. J. Detry, J. D. Durfee, K. A. Jones, N. Martin, Comparison of open-source linear programming solvers, Tech. Rep. SAND2013-8847.
URL `http://prod.sandia.gov/techlib/access-control.cgi/2013/138847.pdf`

[13] D. Ma, M. Saunders, Solving multiscale linear programs using the simplex method in quadruple precision, Proceedings in Mathematics & Statistics 134.

[14] J. Leppnen, M. Pusa, Burnup calculation capability in the PSG2/Serpent Monte Carlo reactor physics code, Proc. M&C (2009) 3–7.

Appendix A

ADER API

A.1 Preface

This document is intended for developers of SERPENT2 and specifically those developers who wish to modify the ADER portions of SERPENT2. For users of General information about SERPENT2 can be found at the SERPENT2 wiki:
`serpent.vtt.ft/mediawiki/index.php/Main_Page`
While ADER specific help can be found in the ADER user manual.
With regards to citing SERPENT2 and ADER, as stated on the SERPENT2 wiki general reference to SERPENT2 may be provided by - J. Leppanen, M. Pusa, T. Vittanen, V. Valtavirta, T. Kaltiaisenaho. *"The Serpent Monte Carlo code: Status, development, and applications in 2013"*. Ann. Nucl. Energy, **82** (2015) 142 - 150. Reference to ADER may be provided by - D. D. Wooten. *ADER - Advanced Depletion Extension for Reprocessing.* (2019).
ADER makes extensive use of the open-source library, Clp, part of the COIN-OR collection of packages. Supporting documentation for Clp can be found at:
`https://github.com/coin-or/Clp`
Of importance is that Clp is distributed under the Eclipse Public Licence which is not a copy-left licence but a rather forgiving open-source licence. Instructions for installing and linking the Clp libraries can be found in the user's manual.

A.2 Introduction

ADER is a source code extension to SERPENT2. Originally developed by Daniel Wooten at the University of California Berkeley ADER provides four key features to users of SERPENT2 - the ability to define relationships between isotopes, elements, and chemicals in a SERPENT2 material; the ability to define how the composition of these SERPENT2 materials may be adjusted; a solution for the optimal material composition and adjustment schedule; and the incorporation of these material adjustments into the nuclear burnup solu-

tion as determined by SERPENT2.
In the following sections the inner workings of ADER will be laid out and explained. The functions will first be organized by themes of collective action and then elaborated upon alphabetically. A special section, A.3, will detail a handful of intrinsic SERPENT2 functions, an understanding of which will facilitate such of ADER.
In section A.4 the functions relating to the interpretation of user input will be detailed. Section A.5 will detail those functions that sort and link user input into the necessary data structures. Section A.6 will detail those functions which construct and solve the optimization problem. Section A.7 will detail those functions that handle the burnup solution. Section A.8 will detail those functions which output data to the user. Section A.9 will detail those functions involved with the testing of ADER. Section A.10 will detail some of the methodology behind parallel computation with ADER.
As a final note it needs to be mentioned up front that SERPENT2 makes extensive use of linked-lists and the term "`ptr`". A "`ptr`" in this sense is not a C-style pointer, a variable which contains a memory address, but the index of a monolithic array at which the desired data may be found. For instance, if `NUMBER-OF-CAT-LEGS-PTR = 6` then one would expect that at `CAT-DATA-ARRAY[6]` would be the value "`4`". C-style pointers are referred to simply as pointers.

A.3 Inside SERPENT2

Considering that ADER is a source code modification to the SERPENT2 base a basic understanding of SERPENT2 programming is essential for working in ADER. The following is *not* intended to be a SERPENT2 API nor will it have near as much detail. Rather, this section is meant to introduce the reader to key SERPENT2 structures so that the reader is then prepared to discover more on their own. The first aspect of SERPENT2 to grasp is not a function but an array - in fact it is best of think of this array as an object from the priciples of object-oriented programming. The `WDB`, or **W**rite **D**ata**B**ase, is a monolithic array of doubles. `RDB` is a write-protected cast of `WDB`. `WDB` holds almost all of the program information for SERPENT2 cast as a `double` and formatted as an array of linked lists. The header file, `locations.h`, holds the initial data structure that `WDB` is built from - this is where the initial array location for the first item in each linked list can be found. Additionally, and of less importance, are the following arrays: `PRIVA` is a doubles array for OpenMP data, `BUF` is a short term accumulation array, `RES1` is a doubles array generally for holding results pulled from `BUF`, `RES2` is an array used for memory optimization in the burnup routines as is `RES3` while both are involved with the threaded behavior of SERPENT2, `ASCII` is a `char*` array. An important concept to keep in mind is that the location of data in all of these arrays is described by the data in `WDB`. This data is accessed and manipulated by functions inside of SERPENT2. While C is not an object-oriented language by default SERPENT2 behaves as an object-oriented program in that many of its objects, these data arrays, should only

be acted on by specific methods, functions inside of SEPRENT. Before diving in to these functions it should be noted that many variables in SERPENT2 have some name of "`ptr`" or a varient thereof. These variables do not indicate C-style pointers - variables declared with an "`*`". Rather they are usually an array index at which some data of interest can be found. This is mentioned as the naming convention could cause confusion.

A.3.1 AverageTransmuXS

Info: According to the averaging scheme chosen by the user this function collects the sub-step averaged transmutation cross sections for the isotopes in material mat. These are stored in the PRIVA array. This is thread safe.

Inputs:

- long mat
- double t1
- double t2
- long id

Returns: void

A.3.2 BurnMaterials

Info: Determines which burnup solver will be used for a given mat on a given step.

Inputs:

- long dep
- long step

Returns: void

A.3.3 BurnMatrixSixe

Info: Returns the number of non-zero entries in a material's burnup matrix without any ADER columns or rows incorporated.

Inputs:

- long mat

Returns: long

A.3.4 BurnupCycle

Info: This is the burnup simulation driver. Schedules transport caculations, burnup calculations, and data output.

Inputs: None
Returns: void

A.3.5 CalculateTransmuXS

Info: Calculates current transport sweep transmutation cross sections. These are used by AverageTransmuXS after having been moved by StoreTransmuXS. Thread safe.

Inputs:

- long mat
- long id

Returns: void

A.3.6 GetPrivateData

Info: Retrieves data from PRIVA array, thread safe.

Inputs:

- long ptr
- long id

Returns: double

A.3.7 GetText

Info: Retreives character string from ASCII array.

Inputs:

- long ptr

Returns: char*

A.3.8 MaterialBurnup

Info: Determines the material burnup in MWd/kgHM.

Inputs:

- long mat
- double *Nbos
- double *Neos
- double t1
- double t2
- long ss
- long id

Returns: void

A.3.9 MakeBurnMatrix

Info: Fills a material's burnup matrix with the coefficients from the Batemann equation. Does not handle ADER materials though it is used in a few ADER tests. Thread safe.

Inputs:

- long mat
- long id

Returns: struct ccsMatrix*

A.3.10 NewItem

Info: Takes linked-list root, `root`, in `WDB` and adds data block of size `sz` returning the array index in `WDB` where the zeroth index of the new data block is found. NOT THREAD SAFE.

Inputs:

- long root
- long sz

Returns: long

A.3.11 NextItem

Info: Takes an item in a `WDB` linked-list, `ptr`, and returns the next item in that same linked-list. Thread safe.

Inputs:

- long ptr

Returns: long

A.3.12 PrepareTransportCycle

Info: Clears various data buffers, distributes simulation data. Should be called before any call to TransportCycle().

Inputs: None
Returns: void

A.3.13 PrintDepOutput

Info: Produces depletion output. `(_dep.m files)`

Inputs: None
Returns: void

A.3.14 ProcessMaterials

Info: Handles data initilization for all SERPENT2 materials.

Inputs: None
Returns: void

A.3.15 ReadInput

Info: Parses user input files - calls data intake routines.

Inputs:

- char *inputfile

Returns: void

A.3.16 StoreTransmuXS

Info: Shuffles material isotopic cross sections into data containers for the begining of a cycle, the end of a cycle, and previous cycle data. Called after CalculateTransmuXS but before AverageTransmuXS.

Inputs:

- long mat
- long step
- long type
- long id
- long iter

Returns: void

A.3.17 TestParam

Info: Basic data intake routine checking developer applied limits on inputs. Called by Read-Input and its subroutines.

Inputs:

- char *pname
- char *fname
- long line
- char *val
- long type

Returns: double

A.3.18 TransportCycle

Info: Primary workhorse of SERPENT2. Runs an entire transport cycle from inactive cycles to last batch. Should only be called after PrepareTransportCycle. Not thread safe.

Inputs: None
Returns: void

A.4 ADER Input

All SEPRENT2 input processing begins with ReadInput. From ReadInput a few of the following functions call the remaining necessary functions - all to read in and store user input.

A.4.1 ADERCreateAderCndEntry

Info: Calls various functions to read a conditions table. Not a thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np

Returns: void

A.4.2 ADERCreateAderControlEntry

Info: Calls various functions to read in a control table. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np
- char* word

Returns: void

A.4.3 ADERCreateAderGroupEntry

Info: Calls functions to read in a group definition. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np
- char* word

Returns: void

A.4.4 ADERCreateAderOxidationEntry

Info: Calls functions to read in an oxidation table. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np
- char* word

Returns: void

A.4.5 ADERCreateAderRemovalEntry

Info: Calls functions to read in a removal table. Not thread safe.

Inputs:

- char* fname

- long line
- char** params
- char* pname
- long np
- char* word

Returns: void

A.4.6 ADERCreateAderStreamEntry

Info: Calls various functions to read in an ADER stream entry. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np

Returns: void

A.4.7 ADERReadAderCndCntData

Info: Reads in the control table data from a conditions block.
Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.8 ADERReadAderCndData

Info: Calls various functions to read in the sub-component pieces of a conditions table. Not thread safe.

Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname
- long np

Returns: void

A.4.9 ADERReadAderCndOptData

Info: Reads in opt entry for conditions table. Not thread safe.

Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.10 ADERReadAderCndOxiData

Info: Reads in oxi entry for a conditions table. Not thread safe.

Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.11 ADERReadAderCndPresData

Info: Reads in a pres entry for a conditions table. Not thread safe.

Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.12 ADERReadAderCndRngData

Info: Reads in a rng entry for a conditions table. Not thread safe.

Inputs:

- char* fname

- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.13 ADERReadAderCndRtoData

Info: Reads in a rto entry for a conditions table. Not thread safe.

Inputs:

- char* fname
- long cnd_ptr
- long j
- long line
- char** params
- char* pname

Returns: long

A.4.14 ADERReadAderControlData

Info: Reads in the elements of a control table. Not thread safe.

Inputs:

- char* fname
- long control_ptr
- long j
- long line
- long np

- char** params
- char* pname

Returns: void

A.4.15 ADERReadAderGroupData

Info: Reads in a group entry. Not thread safe.

Inputs:

- char* fname
- long group_ptr
- long j
- long line
- long np
- char** params
- char* pname

Returns: void

A.4.16 ADERReadAderGroupIsosData

Info: Reads in isotopic data for an element in a group entry. Not thread safe.

Inputs:

- long comp_ptr
- char* fname
- long group_ptr
- long j
- long line
- char** pname
- char* params
- int num_isos

Returns: long

A.4.17 ADERReadAderGroupItemData

Info: Reads in an element entry for a group. Not thread safe.

Inputs:

- char* fname
- long group_ptr
- long j
- long line
- char** params
- char* pname
- long np

Returns: long

A.4.18 ADERReadAderKMaxData

Info: Reads in the maximum k target. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np

Returns: void

A.4.19 ADERReadAderKMinData

Info: Reads in the minimum k target. Not thread safe.

Inputs:

- char* fname

- long line
- char** params
- char* pname
- long np

Returns: void

A.4.20 ADERReadAderNegAdens

Info: Reads the `ader_neg_adens` flag. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np

Returns: void

A.4.21 ADERReadAderTransIterData

Info: Reads the `ader_set_neg_adens` flag. Not thread safe.

Inputs:

- char* fname
- long line
- char** params
- char* pname
- long np
- long k

Returns: long

A.4.22 ADERSetMatAderMem

Info: Initilizes a material's ADER memory block upon the keyword "ader". Not thread safe.

Inputs:

- long loc0
- char** params
- long np
- char* pname
- char* fname
- long line

Returns: void

A.5 ADER Setup

Following the intake of user data, primarily controlled through `ReadInput`, `ADERProcessAderMainData` is called by `main` during the setup portion of the run. ADER functions with the prefix `ADERProcessMaterialAder`... generally refer to setup on a per-material basis of linking the system wide `ADER` data with the individual materials. ADER functions with the prefix `ADERProcessMaterial`... generally refer to setup on a per-material basis of material-wide `ADER` data while those functions with the prefix `ADERProcessAder`... generally refer to setup on an ADER-wide basis. `ADERProcessMaterialAderData` is the primary function which calls the material specific setup functions, it is called from `ProcessMaterials`. It should be noted that throughout the code the terms `reprocessing` and `removal` are used interchangeably in reference to the `removal` table structure - as are their short-hands `remv` and `repro` respectively. With regards to the data structure in ADER a few terms should be elaborated upon. The following definitions are to be taken as the primary definition of a term unless otherwise specified. "ADER data" refers to the section of WDB accessed through the `DATA_PTR_ADER| ptr`. "Material ADER data" refers to the data accessed for each material through the `MATERIAL_ADER_DATA ptr`. "Group isotopes" and "stream isotopes" refer to the isotopes which belong to a given group or stream and which are accessed through the `ADER_MAT_CMP_ISOS_PTR| ptr` and the `ADER_MAT_STREAM_ISOS_PTR| ptr` respectively. "Material isotopes" refer to the isotope list in a material accessed through the `MATERIAL_PTR_COMP ptr` while "material ader isotopes" refers to the isotope list in a material accessed through the `ADER_MAT_ISOS_PTR ptr`. "Material ADER data" refers to the ADER related data for a material stored with that material and accessed through the `MATERIAL_ADER_DATA ptr`. "Composition groups" or "comp groups" refers to groups which have a restricting effect on a material through either the `rng` or `rto` structures.

A.5.1 ADERAddClusterMember

Info: Adds cluster member to the cluster.
Inputs:

- long ader_cluster
- char* ader_strm_mem_id
- double ader_strm_mem_id_index

Returns: void

A.5.2 ADERCheckMaterialClusterIsotopes

Info: Loops through a cluster parent's cluster members calling `ADERMatchMaterialClusterIsotopes`.

Inputs:

- long mat

Returns: void

A.5.3 ADERCheckMaterialRemovalTables

Info: This function loops through all the streams of a material looking for removal table type streams. If no stream was passed in originally this function calls itself once it has found a removal table type stream. If one is found `ADERCompareMaterialRemovalTables` is called.

Inputs:

- long mat
- long passed_ader_mat_stream

Returns: void

A.5.4 ADERCompareMaterialRemovalTables

Info: Loops through the isotopes of two streams ensuring that none of them match. This is intended to be used with removal table type streams for which no isotopes should be shared by removal table type streams in the same material.

Inputs:

- long ader_mat_stream
- long mat
- long passed_ader_mat_stream

Returns: void

A.5.5 ADERFindShadowStream

Info: Searches for a matching stream in a material that is not the passed in material. This finds shadow streams and returns the negative index of this stream as destination streams receive the `ptr` to their source shadow stream as -1 * index-of-source. Source side streams receive unmodifed `ptr`s to their destination shadow streams.
Inputs:

- long ader_mat_stream
- long mat

Returns: long

A.5.6 ADERFindShadowStreamSumStreams

Info: A recursive function capable of giving all summation streams in a summation stream, as deeply nested as the user desires, `ptr`s to their shadow streams.
Inputs:

- long ader_mat_search_stream
- long ader_mat_stream
- long mat

Returns: void

A.5.7 ADERLinkMaterialGroupIsotopes

Info: Loops through the group-style isotope list that is passed in and gives these isotopes `ptr`s to the corresponding `ADER_MAT_ISO` entry.

Inputs:

- long ader_mat_ent_iso
- long mat

Returns: void

A.5.8 ADERLinkMaterialIsotopeIndices

Info: Goes through a material's composition groups and streams calling functions to link their isotopes to the material ader isotopes.

Inputs:

- long mat

Returns: void

A.5.9 ADERLinkMaterialStreamIsotopes

Info: Calls `ADERLinkMaterialGroupIsotopes` for the stream's isotopes and then loops through any summation groups calling itself on these summation groups.

Inputs:

- long ader_mat_strm
- long mat

Returns: void

A.5.10 ADERMatchMaterialClusterIsotopes

Info: Loops through the isotopes of a cluster member and the cluster parent ensuring that all cluster members have the same isotopics.

Inputs:

- long mat

Returns: void

A.5.11 ADERMergeClusters

Info: Merges two ADER clusters together - deprecated the half of the merge which possessed the source side of the stream which led to the merge. Not thread safe.

Inputs:

- long ader_strm_dest_cluster
- long ader_strm_src_cluster

Returns: void

A.5.12 ADERProcessAderClusterMems

Info: Assigns SERPENT2 materials to an ADER cluster based on ADER stream information passed from `ADERProcessAderClusters`. Not thread safe.

Inputs:

- long ader_strm
- long ader_strm_dest_cluster
- char* ader_strm_dest_id
- long ader_strm_src_cluster
- char* ader_strm_src_id

Returns: void

A.5.13 ADERProcessAderClusters

Info: Loops through all ADER streams passing off the stream's material connections to `ADERProcessAderClusterMems`.

Inputs: None
Returns: void

A.5.14 ADERProcessAderGroupFractions

Info: Called by `ADERProcessAderGroups` to normalize user-input group fractions to 1. Not thread safe.

Inputs:

- long grp

Returns: void

A.5.15 ADERProcessAderStreamSourcesAndDests

Info: Replaces null refrences for stream source and destination with textual reference to "NULL". Not thread safe.

Inputs:

- long ader_strm

Returns: void

A.5.16 ADERProcessAderSumGroup

Info: Give's summation groups `ptr`s to the groups which make up their sum. Not thread safe.

Inputs:

- long grp

Returns: void

A.5.17 ADERProcessAderGroups

Info: Calls functions to normalize group fractions to 1 and to link sumamtion groups to the groups which compose their sum. Loops through all ADER groups. Not thread safe.

Inputs: None
Returns: void

A.5.18 ADERProcessAderMainData

Info: Sets default ADER transport loop iteration maximum if the user has not provided a value. Calls functions to setup data for ADER groups, streams, and clusters. Not thread safe.

Inputs: None
Returns: void

A.5.19 ADERProcessAderStreams

Info: Loops through all ADER streams calling `ADERProcessAderStreamSourcesAndDests`. Not thread safe.

Inputs: None
Returns: void

A.5.20 ADERProcessMaterialAderClusterMems

Info:
Inputs:

- long ader_cluster
- long mat_ader_data

Returns: void

A.5.21 ADERProcessMaterialAderClusterParent

Info: Gives a material a `ptr`s to its cluster parent material.
Inputs:

- long ader_cluster
- long mat_ader_data

Returns: void

A.5.22 ADERProcessMaterialAderClusters

Info: Loops through ADER clusters and calls functions to assign materials to clusters. Not thread safe.

Inputs:

- long mat

Returns: void

A.5.23 ADERProcessMaterialAderCndCntData

Info: Loops through ADER control tables to give `mat` a `ptr` to the control table its condition table designates. Not thread safe.

Inputs:

- long ader_cnd_cnt
- long mat

Returns: void

A.5.24 ADERProcessMaterialAderCndData

Info: Calls functions to create material group entries and to link range and ratio restrictions to these groups. Not thread safe.

Inputs:

- long ader_cnd_ent

- char *ader_type
- long mat

Returns: void

A.5.25 ADERProcessMaterialAderCndOptData

Info: Attaches optimization data to materials from ADER condition tables. Not thread safe.

Inputs:

- long ader_cnd_opt
- long mat

Returns: void

A.5.26 ADERProcessMaterialAderCndOxiData

Info: Attaches oxidation table data to a material. Not thread safe.

Inputs:

- long ader_cnd_oxi
- long mat

Returns: void

A.5.27 ADERProcessMaterialAderCndPresData

Info: Attaches preservation data from ADER conditions tables to materials. Not thread safe.

Inputs:

- long ader_cnd_pres
- long mat

Returns: void

A.5.28 ADERProcessMaterialAderData

Info: Loops through all SERPENT2 materials checking for those material's under ADER control. First calls functions to add materials to ADER clusters, to ensure that all cluster members have matching isotopics, to add conditions to materials, to connect streams to materials, and to link various `ptr`s related to isotopes and their data. Following these functions shadow streams are processed as all materials must already have streams to find the shadows (see the user's manual for this term). The optimization entries are added to materials and then, following a check on some user data the material cluster composition optimization matrices are first constructed and allocated space in another function call. Once space has been allocated these matrices are then filled for the first time. Not thread safe.

Inputs: None
Returns: void

A.5.29 ADERProcessMaterialAderIsosData

Info: Creates ADER iso entry for every iso in a material. Not thread safe.

Inputs:

- long mat

Returns: void

A.5.30 ADERProcessMaterialClusterOptEntry

Info: Copies the optimization target in a cluster to the cluster-parent material.

Inputs:

- long mat

Returns: void

A.5.31 ADERProcessMaterialCndGroupData

Info: If no group data exists for the given group in the given material creates an entry in that material for that group and calls functions to process the group's composition. Not thread safe.

Inputs:

- char *ader_grp_id

- long mat

Returns: long

A.5.32 ADERProcessMaterialCndRngData

Info: Copies range data from ADER conditions table to material condition table.
Inputs:

- long ader_cnd_rng
- long ader_mat_cmp
- long mat

Returns: void

A.5.33 ADERProcessMaterialCndRtoData

Info: Copies ratio restrictions from ADER control tables to material control tables. Not thread safe.

Inputs:

- long ader_cnd_rto
- long ader_mat_cmp
- long mat

Returns: void

A.5.34 ADERProcessMaterialConditions

Info: Builds a material's conditions block from the designated ADER conditions block. Calls various function to fill in the pieces. Not thread safe.

Inputs:

- long mat

Returns: void

A.5.35 ADERProcessMaterialGroupComposition

Info: Copies data from ADER groups to the material groups - whether those are streams or condition groups. Not thread safe.

Inputs:

- long ader_grp
- long ader_mat_ent_ele_ptr
- long ader_mat_ent_iso_ptr
- long ele_iso_fix_check
- long mat
- long stream_check

Returns: void

A.5.36 ADERProcessMaterialRemovalData

Info: Sets up removal-table type stream for a material adding the stream to the material's stream list and calling functions to fill in the stream data. Not thread safe.

Inputs:

- long ader_mat_strm
- long ader_strm
- long mat

Returns: void

A.5.37 ADERProcessMaterialRemovalEle

Info: Loops through elements in a material's removal table type stream and creates and fills in these elements' isotopic information. Not thread safe.

Inputs:

- long ader_mat_stream
- long mat

Returns: void

A.5.38 ADERProcessMaterialRemovalEntryData

Info: Loops through entries of an ADER removal table, being added to a material stream, sorting the entries into elemental and isotopic data lists in the stream as apporpiate. Not thread safe.

Inputs:

- long ader_mat_strm
- long ader_rem
- long ele_iso_fix_check
- long mat

Returns: void

A.5.39 ADERProcessMaterialRemovalIsos

Info: Goes through newly created stream isotopes for a stream based of a removal table and links these isotopes to the stream element's they should be linked to. Creates elements if they need creating. Not thread safe.

Inputs:

- long ader_mat_stream
- long mat

Returns: void

A.5.40 ADERProcessMaterialShadowStreamCompMatrixSection

Info: Copies a stream's composition matrix index information to the source side stream.

Inputs:

- long ader_mat_stream

Returns: void

A.5.41 ADERProcessMaterialShadowStreams

Info: Loops through a material's streams calling functions to give these streams `ptrs` to their shadow streams.
Inputs:

- long mat

Returns: void

A.5.42 ADERProcessMaterialStreamData

Info: Copies data from an ADER stream to the material stream.
Inputs:

- long ader_mat_strm
- long ader_strm
- long mat

Returns: void

A.5.43 ADERProcessMaterialStreamGroupData

Info: Creates data space for storing a stream's burnup data. Also calls functions to fill in a stream's elemental and isotopic data. Calls functions to process summation stream data. Not thread safe.

Inputs:

- long ader_grp
- long ader_mat_strm
- long ele_iso_fix_check
- long mat

Returns: void

A.5.44 ADERProcessMaterialStreams

Info: Loops through all ADER streams assigning them to `mat` if they connect to that material. Not thread safe.

Inputs:

- long mat

Returns: void

A.5.45 ADERProcessMaterialStreamUnFixedEle

Info: Loops through a material's isotopes creating a stream element isotope entry for that isotope if if does not yet have one in its host stream element. Not thread safe.

Inputs:

- long ader_mat_stream_ele
- long ader_mat_stream_iso_ptr
- long mat

Returns: void

A.6 ADER Optimization

In the third loop over materials as seen in `ADERProcessMaterialAderData`, `ADERCreateMaterialCompMatrix` is called for each material. This function and its associated calls allocate the space for the material optimization matrix as well as setting up the associated meta-data - mostly in the form of `ptr`s. It should be noted that the material optimization matrix is incredibly sparse - more than 99% empty space. An early, and most likely misguided, design decision led to this matrix being created and stored in full. For a single SERPENT2 material with all isotopes being tracked about 6 GB of space is consumed by a single matrix. It is entirely possible, albeit with a significant architectural overhaul, to forgo storing this matrix and rather generate it on the fly. This IS NOT how the composition matrix is currently handled. Rather this information is included as a note on why the memory footprint of ADER is so high and as a guidepost for future developers.
Following the creation of this sparse matrix the final loop over materials in `ADERProcessMaterialAderData` calls `ADERFillMaterialCompMatrix` which will then fill in the static data for the material optimization matrix. With regard to many of the `ADERFill...` functions which take a SERPENT `mat` as an input, many of these functions have restrictions and limitations on what kinds of `mat`s can be passed to them. Many

of these restrictions are not enforced by code checks - rather they are elaborated in the comments of these functions. An important note is that at various points in the code this matrix, which represents the linear programming problem of optimizing the material composition, is referred to as both the "optimization matrix", the "composition matrix" and various combinations thereof, including the abbreviations "opt matrix" and "comp matrix" respectively. In this case "comp" should not be confused with "cmp" the latter of which is used to refer to a conditions entry involving a group.
Following the initial filling of the static optimization data the program progresses into the burnup cycles as regulated by `BurnupCycle` mentioned in section A.3.
`ADERCorrectTransportCycle` is called after the first transport cycle of each burnup step. This function, and its associated calls, will fill in the dynamic data for the material optimization matrices, solve these matrices, incorporate any instantaneous changes, and re-run the transport cycle checking to ensure the neutron multiplication factor, k_{eff}, is in the target bounds. If k_{eff} is, the program proceeds to building and solving the depletion problem. If not and iterations remain `ADERCorrectTransportCycle` runs through the entire process again. The key data gained from all of these steps just mentioned are the stream values, as determined by ADER, that will push the material compositions towards their optimum states. As a final note throughout many functions a variable by the name of `adj` can be seen in many places incorporated into if-statements which could dramatically alter program flow. As of V.1.0 the `adj` portion of the algorithms is unused - its value is always set to 0. This variable exists as part of an early development effort to create algorithmic space for an interaction scheme inside of the ADER routines. As no such scheme was implemented `adj` hangs on as a vestigial organ.

A.6.1 ADERAllocateClpMemory

Info: Allocates system memory for the CLP process. Returns a vector of pointers to these memory allocations.

Inputs:

- long ader_mat_matrix_data

Returns: double**

A.6.2 ADERAverageValue

Info: Using the SERPENT2 weights for averaging returns an average of the values passed in over the time interval specified. By using the SERPENT2 weights differint averaging schemes can be employed by SERPENT2 and will be passed down to this functino as well through the `WDB` structure.

Inputs:

- double bos_value
- double eos_value
- double ps1_value
- double t1
- double t2
- long dep

Returns: double

A.6.3 ADERBuildClpModel

Info: Fills in the CLP problem data into the arrays pointed to. Returns the number of non-zero matrix entries in the CLP problem. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- double *column_lower_bounds
- double *column_upper_bounds
- double *index_column_starts
- double *objective_row
- double *row_lower_bounds
- double *row_indices
- double *row_upper_bounds
- double *values

Returns: long

A.6.4 ADERBurnMaterials

Info: Orchestrates the solution of the burnup problem for an ADER cluster. Distributes this solution and calls various post-burnup checks and updates, such as `MaterialBurnup` and `UpdateCIStop`.

Inputs:

- long burn_ci_flag
- long mat
- long mode
- long num_sub_steps
- long step
- long type

Returns: void

A.6.5 ADERClearAderXSData

Info: Loops through all materials clearing all material ADER isotope CUR cross sections.

Inputs: None
Returns: void

A.6.6 ADERClearMaterialCompMatrixClusterMemPresRowBounds

Info: Sets preservation row bounds in a material's comp matrix to zero. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long mat

Returns: void

A.6.7 ADERClearPropStreamAmts

Info:
Inputs:

- long adj
- long mat

Returns: void

A.6.8 ADERClearTargetPropStreamAmts

Info:
Inputs:

- long ader_mat_stream
- long adj

Returns: void

A.6.9 ADERCopyMaterialFlux

Info: Stores the material flux passed in. Not thread safe.

Inputs:

- double flx
- double flx_new_avg
- double flx_old_avg
- long mat

Returns: void

A.6.10 ADERCorrectTransportCycle

Info: Called from `BurnupCycle`. Called after an initial transport sweep executed by `TransportCycle`. Orders the filling-in of the material composition matrices, their solution, and the incorporation of any instentaneous changes. Following the incorporation of such changes their effects on reactivity are assessed with a new transport sweep. After which, if the multiplication factor of the system is within the user bounds or if the number of iterations have been exceeded, this function will exit.
Inputs:

- long dep
- long step

Returns: void

A.6.11 ADERCountStream

Info: Counts the number of streams belonging to the passed in stream - this number will only be greater than 1 for summation streams. Gives the stream and all its children their burn matrix index.

Inputs:

- long ader_mat_stream
- long ader_mat_stream_count
- long num_rows

Returns: long

A.6.12 ADERCountStreamIsos

Info: Counts the number of isotopes with non-zero fraction in a stream and all its children. Returns `num_non_zero_ents` incremented by this count.

Inputs:

- long ader_mat_stream
- long num_non_zero_ents

Returns: long

A.6.13 ADERCreateMaterialClusterMemCompMatrixSection

Info: Orchestrates the building of the composition matrix for the portion coming from `ader_mat_cluster_ent_mem`. Not thread safe.

Inputs:

- long ader_mat_cluster_ent_mem
- long ader_mat_matrix_data

Returns: void

A.6.14 ADERCreateMaterialCmpGroupCompMatrixSection

Info: Creates columns in the composition matrix for the passed in `cmp` group as well as any rows for any ratios this group may be involved with. Handles summation rows as well. Not thread safe.

Inputs:

- long ader_mat_cmp
- long ader_mat_matrix_data

Returns: void

A.6.15 ADERCreateMaterialCompMatrix

Info: Loops through cluster members of the cluster with parent of `mat` calling `ADERCreateMaterialClusterMemCompMatrixSection` to create each cluster member's compositional matrix section. Not thread safe.

Inputs:

- long mat

Returns: void

A.6.16 ADERCreateMaterialCompMatrixCol

Info: Adds a column to the material compositional matrix. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- double col_lower_bound
- double col_upper_bound

Returns: long

A.6.17 ADERCreateMaterialCompMatrixRow

Info: Creates a new row in a material composition matrix. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- double row_lower_bound
- double row_upper_bound

Returns: long

A.6.18 ADERCreateMaterialEleCompMatrixSection

Info: Loops through a material's elements creating the associated matrix rows and columns as needed and setting initial column and row bounds. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.19 ADERCreateMaterialIsoCompMatrixSection

Info: Loops through a material's ADER isotopes creating the associated columns and rows in the material's composition matrix. Also sets initial bounds for these columns and rows. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.20 ADERCreateMaterialOxiCompMatrixSection

Info: Creates the oxidation row in a material's composition matrix as well as setting the initial bounds. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long ader_mat_oxi

Returns: void

A.6.21 ADERCreateMaterialPresCompMatrixSection

Info: Creates a row for the preservation entry of a material. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long ader_mat_pres

Returns: void

A.6.22 ADERCreateMaterialRhoCompMatrixSection

Info: Creates a row for reactivity control in a material's composition matrix. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.23 ADERCreateMaterialStreamCompMatrixSection

Info: Creates columns and rows for a material's stream's composition matrix section. Calls itself to deal with summation streams. Not thread safe.

Inputs:

- long ader_mat_matrix_data
- long ader_mat_stream

Returns: void

A.6.24 ADERDeallocateTarget

Info: Frees the memory associated with the passed in array of length `target_size`. NOT THREAD SAFE.

Inputs:

- double **target
- long target_size

Returns: void

A.6.25 ADERFillMaterialClusterMemCompMatrixSection

Info: Similar to `ADERCreateMaterialClusterMemCompMatrixSection` this function calls helper functions to fill in the problem data to a material's composition matrix.

Inputs:

- long ader_mat_cluster_ent_mem
- long ader_mat_matrix_data

Returns: void

A.6.26 ADERFillMaterialCmpGroupCompMatrixSection

Info: Calls functions to fill in the group fractions for a comp group's elemental and isotopic specifications. Calls functions to fill in ratio and summation information for a group.
Inputs:

- long ader_mat_cmp
- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.27 ADERFillMaterialCmpRtoCompMatrixSection

Info: Fills in composition matrix data to describe a limited ratio between two comp groups.

Inputs:

- long ader_mat_cmp_rto
- long ader_mat_matrix_data
- long ader_mat_matrix_first_col_id
- long mat_ader_data

Returns: void

A.6.28 ADERFillMaterialCmpSumCompMatrixSection

Info: Fills in composition matrix data to describe the relationships between a comp group and its summation groups.

Inputs:

- long ader_mat_cmp
- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.29 ADERFillMaterialCompMatrix

Info: Loops through the members of a material cluster calling `ADERFillMaterialClusterMemCompMatrixSection` to fill in their composition matrix contributions.

Inputs:

- long mat

Returns: void

A.6.30 ADERFillMaterialCompMatrixEleData

Info: Loops through the passed in list of elements, `ader_mat_ent_ele`, as taken from a comp group or stream and fills in the elemental fraction data into the material composition matrix.

Inputs:

- long ader_mat_ent_ele
- long ader_mat_matrix_col_id
- long ader_mat_matrix_data
- long mat_ader_data
- double mult
- long sign
- long type

Returns: void

A.6.31 ADERFillMaterialCompMatrixIsoData

Info: Loops through the passed in isotope list, `ader_mat_ent_iso`, from a stream or composition group filling in the isotopic fraction data into the composition matrix.

Inputs:

- long ader_mat_ent_iso
- long ader_mat_matrix_col_id
- long ader_mat_matrix_data
- long mat_ader_data
- double mult
- long sign
- long type

Returns: void

A.6.32 ADERFillMaterialCompMatrixObjRow

Info: Calls functions to fill in the material composition matrix objective row given the objective target.

Inputs:

- long mat

Returns: void

A.6.33 ADERFillMaterialEleCompMatrixSection

Info: Loops through a material's elements filling in their data to the material composition matrix.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.34 ADERFillMaterialIsoCompMatrixSection

Info: Loops through a material's ADER isotopes filling in their composition matrix data.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.35 ADERFillMaterialObjActFeedAndRemvCompMatrixSection

Info: Fills in optimization data for feed and removal streams as the target by looping through all cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.36 ADERFillMaterialObjActFeedCompMatrixSection

Info: Fills in optimization data for feed streams as the target looping through all cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.37 ADERFillMaterialObjActReacCompMatrixSection

Info: Fills in optimization data for reac streams as the target by looping through all cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.38 ADERFillMaterialObjActRedoxCompMatrixSection

Info: Fills in optimization data for redox streams as the target by looping through the cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.39 ADERFillMaterialObjActRemvCompMatrixSection

Info: Fills in optimization data for remv streams as the target by looping through the cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.40 ADERFillMaterialObjActStreamsCompMatrixSection

Info: Fills in optimization data for all streams as the target by looping through the cluster members of cluster parent `mat`.

Inputs:

- long mat

Returns: void

A.6.41 ADERFillMaterialObjActTransfersCompMatrixSection

Info: Fills in optimization data for streams which transfer substances between SERPENT2 materials, of which are located in the cluster who's parent is `mat` and who's members will be loops through to find such streams.

Inputs:

- long mat

Returns: void

A.6.42 ADERFillMaterialObjGrpCompMatrixSection

Info: Fills in optimization data for target group found in the cluster members of cluster parent `mat`. These members are looped through to find all refernces to the target group.

Inputs:

- char* ader_mat_opt_target
- long mat

Returns: void

A.6.43 ADERFillMaterialObjStreamCompMatrixSection

Info: Fills in optimization data for a specific target stream. All instances of which will be found by looping through the cluster members of cluster parent `mat`.

Inputs:

- char* ader_mat_opt_target
- long mat

Returns: void

A.6.44 ADERFillMaterialOxiCompMatrixSection

Info: Loops through the ADER elements found in `mat_ader_data` filling in their data to any oxidation rows in the composition matrix.

Inputs:

- long ader_mat_matrix_data
- long mat_ader_data

Returns: void

A.6.45 ADERFillMaterialPresCompMatrixSection

Info: This function is more of a placeholder for future developers to add more `pres` options. As of V.1.0 there is only one option and so this function calls `ADERFillMaterialPresMolsCompMatrixSection`.

Inputs:

- long ader_mat_cluster_ent_mem
- long ader_mat_matrix_data
- long ader_mat_pres

Returns: void

A.6.46 ADERFillMaterialPresMolsCompMatrixSection

Info: Loops through a material's streams incorporating their data into the preservation row of the composition matrix.

Inputs:

- long ader_mat_cluster_ent_mem
- long ader_mat_matrix_data
- long ader_mat_pres

Returns: void

A.6.47 ADERFillMaterialStreamCompMatrixSection

Info: Fills in composition matrix data for the given stream and any summation streams it may have via a recurssive call.

Inputs:

- long ader_mat_cluster_ent_mem
- long ader_mat_matrix_data
- long ader_mat_stream

Returns: void

A.6.48 ADERGetEigenBias

Info: Calculates the absorption probability of a reactive material under ADER control.

Inputs:

- long dep
- long mat

- double t1
- double t2

Returns: double

A.6.49 ADERGetIsoBurnMatrixIndex

Info: Returns the burn matrix index for the specified isotope. Thread safe.

Inputs:

- char* func
- long mat
- long nuc

Returns: long

A.6.50 ADERGetLeakageCorrectionFactor

Info: Calculates and stores the non-leakage probability. Thread safe.

Inputs:

- long dep
- long i
- long step

Returns: void

A.6.51 ADERGetMatEleIsoFrac

Info: Updates elemental fractions and isotopic elemental fractions for all elements and material ADER isotopes in a material.

Inputs:

- long mat

Returns: void

A.6.52 ADERGetMaterialCompMatrixElement

Info: A utility function for printing and testing. Returns the value found at the specified indices in the given composition matrix. Thread safe.

Inputs:

- long ader_mat_matrix_data
- long col_index
- long row_index

Returns: double

A.6.53 ADERGetMaterialRemovalAmounts

Info: Loops through a material's streams calling `ADERGetStreamRemovalAmounts` on any `rem` type stream.

Inputs:

- long mat
- long i
- double t1
- double t2

Returns: void

A.6.54 ADERGetMaterialShadowStreamIsoFracs

Info: Provides destination shadow stream isotopes with their proper fractions as the relative fractions for these isotopes come not from the material hosting the destination side stream but from the material hosting the source side stream.

Inputs:

- long ader_mat_stream

Returns: void

A.6.55 ADERGetMaterialStreamUnFixedEleIsoFracs

Info: Determines isotopic fractions of elements for a stream with elements who's isotopic composition is not specified.

Inputs:

- long ader_mat_matrix_data
- long ader_mat_stream
- long mat

Returns: void

A.6.56 ADERGetStreamRemovalAmounts

Info: Determines and stores the amount of material moved by a `rem` type stream.
Inputs:

- long ader_mat_stream
- long mat
- long mat_ader_data
- double t1
- double t2

Returns: void

A.6.57 ADERGetTransportInformation

Info: Calls functions to generate and store needed information post transport-sweep.

Inputs:

- long dep
- long i
- long step

Returns: void

A.6.58 ADERMoveBosEosPs1Values

Info: Shuffles values amoung the specified indices to match SERPENT2 averaging.
Inputs:

- long avg_index
- long cur_index
- long bos_index
- long eos_index
- long ps1_index
- long mat
- long step
- long iter

Returns: void

A.6.59 ADERMoveCrossSection

Info: Loops through a material's ADER isotopes calling `ADERMoveBosEosPs1Values` for an isotopes various cross sections.

Inputs:

- long mat
- long step
- long iter

Returns: void

A.6.60 ADERNormalizeCrossSection

Info: Loops through a material's ADER isotopes normalizing their cross sections by `flx`.

Inputs:

- double flx
- long mat

Returns: void

A.6.61 ADEROperateMaterial

Info: If any one function could be considered the workhorse of ADER, it would be this one. This function, called from `ADERCorrectTransportCycle` orders and executes the updating of a cluster's composition matrix as well as the determination of the optimal solution. Numerous components of the composition matrix require updating with each ADER iteration. The majority of these updates are due to changing isotopic compositions and possible changes to material densities. The `stream_counter` variable is necessary as ADER simulations with only prescriptive streams will have no "optimal" solution and as such the solving of such a problem should be skipped. Once the various updates to the composition matrix have been completed the solver function, `ADEROperateMaterialCompMatrix` is called.

Inputs:

- long adj
- long dep
- long i
- long mat
- long step
- double t1
- double t2

Returns: void

A.6.62 ADEROperateMaterialCompMatrix

Info: Manages the construction, solution, and distribution of solution data for the optimization problem.

Inputs:

- long adj
- long i
- long mat
- long step

Returns: void

A.6.63 ADERParseClpSolution

Info: Manages the distribution of the optimization solution to the various comp groups and streams.

Inputs:

- long adj
- long i
- long mat
- long step
- double *solution

Returns: void

A.6.64 ADERParseStreamClpSolution

Info: Recurssive function for distributing optimization solution data to streams, which may themselves have summation streams.

Inputs:

- long ader_mat_stream
- long adj
- double *solution

Returns: void

A.6.65 ADERProcessMaterialDiscStreamEffects

Info: Updates material isotopics and density for discrete stream transfers.

Inputs:

- long ader_mat_stream
- long adj
- long i
- long mat

Returns: void

A.6.66 ADERProcessMaterialShadowStreamEleAndIsoFracs

Info: Manages updating destination side shadow streams with elemental and isotopic fractions from the source side streams.

Inputs:

- long mat

Returns: void

A.6.67 ADERScoreCrossSection

Info: Sorts cross section information obtained from `CalculateTransmuXS` into the appropriate ADER containers.

Inputs:

- long abs
- long E
- long id
- long mat
- long nuc
- long rea
- double value

Returns: void

A.6.68 ADERSetMaterialCompMatrixClusterMemColBounds

Info: Manages the collection of column bounds information and the assignment of these column bounds for a particular cluster member - primarily by calling other functions.

Inputs:

- long mat

Returns: void

A.6.69 ADERSetMaterialCompMatrixClusterMemPresRowBounds

Info: Adjusts preservation row bounds.

Inputs:

- long ader_mat_matrix_data
- long mat
- double value

Returns: void

A.6.70 ADERSetMaterialCompMatrixClusterMemRhoRowEntries

Info: Loops through a material's ader isotopes, if this material is under reactivity control, filling in the composition matrix reactivity data for each isotope.

Inputs:

- long dep
- long mat
- double t1
- double t2

Returns: void

A.6.71 ADERSetMaterialCompMatrixClusterMemRowBounds

Info: Sets elemental and isotopic balance row bounds for `mat`. Loops through the streams of the same calling `ADERSetMaterialCompMatrixClusterMemRemovalTableRowBounds` for rem type streams.

Inputs:

- long dep
- long i
- long mat
- double t1
- double t2

Returns: void

A.6.72 ADERSetMaterialCompMatrixColBounds

Info: A utility function for adjusting a column bound in a composition matrix.

Inputs:

- long bound
- long increment
- long mat_matrix_col_id
- long mat_matrix_data
- double value

Returns: void

A.6.73 ADERSetMaterialCompMatrixElement

Info: A utility function for setting the value of a composition matrix at a given index.

Inputs:

- long index_col
- long index_row
- long ader_mat_matrix_data
- double value

Returns: void

A.6.74 ADERSetMaterialCompMatrixRowBounds

Info: A utility function for setting the row bounds for a row in a composition matrix.

Inputs:

- long bound
- long increment
- long mat_matrix_data
- long mat_matrix_row_id
- double value

Returns: void

A.6.75 ADERSetShadowStreamRemovalAmount

Info: Passes removal amounts from rem type stream elements and isotopes to their destination side shadow stream equivalents.

Inputs:

- long ader_mat_stream
- long ele_id
- long iso_id
- double value

Returns: void

A.6.76 ADERSolveClpModel

Info: Converts the SERPENT2 representation of the optimization problem into the form exepcted by the CLP library. Passes this constructed problem to the CLP library from which this function retreives the problem solution.

Inputs:

- double *column_lower_bounds
- double *column_upper_bounds
- double *index_column_starts
- long num_cols
- long num_ent
- long num_rows
- double *objective_row
- long opt_dir
- double *row_lower_bounds
- double *row_indices
- double *row_upper_bounds
- double *solution

- double *values
- long mat

Returns: void

A.6.77 ADERUpdateMaterialDiscStreamEffects

Info: Loops through a material's streams passing discrete type streams off to `ADERProcessMaterialDiscStreamEffects` so that these stream's changes to the material may be promptly incorporated.

Inputs:

- long adj
- long i
- long mat

Returns: void

A.7 ADER Burnup

Following the solution of the material optimization problem as handled by `ADERCorrectTransportCycle`, `BurnupCycle` will shortly call `BurnMaterials` which will manage the burnup solution for the current step. `BurnMaterials` will then pass off ADER cluster parent materials to `ADERBurnMaterials` which manages the burnup solution for ADER clusters. Much like the composition matrices, material's which are linked by ADER streams must have their burnup matrices solved as a whole. The burnup matrix as produced by ADER is ill-behaved when solved with TTA methods. As such, only the CRAM solution method is used on ADER material burnup problems. The burnup matrix is built one column at a time with no values dropped. It is stored as in a dense column-major format.

A.7.1 ADERBurnMaterials

Info: Orchestrates the solution of the burnup problem for an ADER cluster. Distributes this solution and calls various post-burnup checks and updates, such as `MaterialBurnup` and `UpdateCIStop`.

Inputs:

- long burn_ci_flag

- long mat
- long mode
- long num_sub_steps
- long step
- long type

Returns: void

A.7.2 ADERGetBurnMatrixSizeData

Info: Returns an array of pointers to memory allocations for the burnup problem.
Inputs:

- long mat

Returns: double**

A.7.3 ADERMakeBurnMatrix

Info: This function replicates, through itself and its child-functions, the functionality of `MakeBurnMatrix` with the addition of functionality as needed by ADER. The burnup matrix is stored in a dense column-major format. Cross sections are pulled from `rea` structures which are attached to `nucs`. As such, the sequence of functions `CalculateTransmuXS` `StoreTransmuXS` must be called for each cluster member such that the cross sections for that cluster member are used when filing in the burnup matrix data as the two mentioned functions move the XS information from the material `dep` structures to the `rea` structures.
Inputs:

- struct ccsMatrix burn_matrix
- double* col_vector
- long mat
- long num_ents
- long num_rows
- long step
- long step_type
- double t1

- double t2

Returns: void

A.7.4 ADERMapDensityVector

Info: If used in "receive" mode this function pulls isotopic abundance and proportional stream transfer amounts from the burnup problem solution vector. If used in "send" mode this function fills a vector with isotopic abundance information and continuous stream injection data. Thread safe.

Inputs:

- double* ader_burn_matrix_N
- double* ader_burn_matrix_starts
- long adj
- long direction
- long mat
- long predictor
- double total_time

Returns: void

A.7.5 ADERMapDensityVectorStream

Info: Called by `ADERMapDensityVector` to tunnel into streams and their possible summation streams and thus gather, fill, and send the requsite information.

Inputs:

- double* ader_burn_matrix_N
- long ader_mat_stream
- long adj
- long direction
- double total_time

Returns: void

A.7.6 ADERProcessBurnMatrixContStream

Info: Fills in and stores the column in the burnup matrix corresponding to `ader_mat_stream` and any summation streams it may have.

Inputs:

- struct ccsMatrix* burn_matrix
- long ader_mat_stream
- double* col_vector
- long entry_number
- long mat
- long num_rows
- long* return_array

Returns: long

A.7.7 ADERProcessBurnMatrixFissionYield

Info: Manages fission data and fission yield data for an isotope in the burnup matrix.

Inputs:

- long ader_mat_iso
- double* col_vector
- long fission_yield_data
- long mat
- double mat_flux
- long nuc
- long omp_id
- long rea
- long type

Returns: void

A.7.8 ADERProcessBurnMatrixPropStream

Info: Determines and fills proportional stream contributions to the burnup matrix.

Inputs:

- long ader_mat_ader_iso
- long ader_mat_stream
- double* col_vector
- long mat

Returns: void

A.7.9 ADERProcessBurnMatrixTransmutationAndDecay

Info: Manages contributions from the transmutation and decay of isotopes and their children to the burnup matrix.

Inputs:

- long ader_mat_iso
- double* col_vector
- long mat
- double mat_flux
- long nuc
- long omp_id
- long rea
- long reaction_product_nuc
- long type

Returns: void

A.7.10 ADERStoreBurnMatrixColumn

Info: Stores a column of the burnup matrix into the ccsMatrix structure.
Inputs:

- struct ccsMatrix* burn_matrix
- long col_index
- double* col_vector
- long entry_number
- long num_rows

Returns: long

A.8 ADER Output

Following the solution of the burnup problem program flow will return to `BurnupCycle`. Following each depletion step `BurnupCycle` will call `PrintDepOutput` to reproduce ALL of SERPENT2's output each burnup step. In a loop over materials `PrintDepOutput` will call `ADERPrintOutput`. Also included in this section are several ADER utility functions that have no impact or use for the casual observer - rather many of these functions can be activated via compilation flags to output copious amounts of program data that could be useful for debugging or optimizing.

A.8.1 ADERGetBurnMatrixValue

Info: A utility function for testing and printing. Returns the value of the burnup matrix at the given indices. Thread safe.

Inputs:

- long col_index
- struct ccsMatrix* burn_matix
- long row_index

Returns: double

A.8.2 ADERGetTargetRemovalAmount

Info: Determines the net amount of an isotope or element that is moved by all of a material's `rem` type streams by looping through all of a material's streams and calling `ADERGetStreamTargetRemovalAmount`. Thread safe.

Inputs:

- long mat
- long ele
- long iso

Returns: double

A.8.3 ADERGetStreamTargetRemovalAmount

Info: Returns the amount of a target element or isotope which is moved by `ader_mat_stream`. Thread safe.

Inputs:

- long ader_mat_stream
- long ele
- long iso

Returns: double

A.8.4 ADEROutputBurnMatrixAsCsv

Info: A utility function activated by the compilation flag -DADER_DIAG. Outputs the specified burnup matrix as an ASCII compliant csv file. Thread safe.

Inputs:

- struct ccsMatrix* burn_matrix
- long ader_mat_burn_matrix_num_rows
- long mat
- long step
- long sub_step

Returns: void

A.8.5 ADEROutputMaterialCompMatrixAsCsv

Info: Utility function activated by the compliation flag DADER_DIAG. Outputs the given composition matrix as a csv file. Thread safe.

Inputs:

- long ader_mat_matrix_data
- long cluster_num

Returns: void

A.8.6 ADEROutputMaterialCompMatrixData

Info: A utility function activated with the compilation flag -DADER_DIAG. Outputs a json formatted file with a complete description of the ADER environment.

Inputs: None
Returns: void

A.8.7 ADEROutputMaterialCompMatrixStreamData

Info: A utility function to tunnel into streams and their summation streams for `ADEROutputMaterialCompMatrixData` and put their data into the json file.

Inputs:

- long ader_mat_cluster_mem
- long ader_mat_stream
- long ader_mat_stream_sum_stream_check
- FILE* fp
- long level
- long tab_level
- long tab_length

Returns: long

A.8.8 ADERPrintCrossSections

Info: A utility function which outputs material ader isotope cross sections to file.

Inputs:

- long dep
- long i
- long mat
- long step
- double t1
- double t2

Returns: void

A.8.9 ADERPrintFinalStepCrossSections

Info: Utliity function which outputs material ader isotope cross sections AFTER the conclusion of ADER iterations. This is such that the ADER cross sections have been updated with the latest material flux whereas in `ADERPrintCrossSections` the ADER cross sections reported are those the isotopes had before the most recent discrete streme adjustment.

Inputs:

- long dep
- long mat
- long step
- double t1
- double t2

Returns: void

A.8.10 ADERPrintIndentedOutput

Info: Utility function to ease tab-setting in json files.

Inputs:

- FILE* fp
- char* print_data
- long tab_length
- long tab_level

Returns: void

A.8.11 ADERPrintListsHierarchy

Info: A utliity function activated by -DADER_DIAG which outputs to a text file a summary of ADER `WDB` address information.

Inputs: None
Returns: void

A.8.12 ADERPrintMaterialStreamIsotopes

Info: Utility function activated by -DADER_DIAG. Called by `ADERPrintListsHierarchy` to tunnel into stream and summation stream data.

Inputs:

- long ader_mat_stream
- FILE *fp
- long mat

Returns: void

A.8.13 ADERPrintOutput

Info: NOT A UTILITY. Prints to the "_dep.m" output file ADER problem information associated with `burn_mat`. Calls `ADERPrintOutputStreamData` to tunnel into streams and any summation streams and output their data.

Inputs:

- long burn_mat
- FILE* fp
- char* mat_name

Returns: void

A.8.14 ADERPrintOutputStreamData

Info: Called by `ADERPrintOutput` to facilitate printing of ADER stream data into the "_dep.m" file.

Inputs:

- long ader_mat_stream
- FILE* fp
- char* mat_name

Returns: void

A.8.15 ADERPrintSumStreams

Info: Utility function called by `ADERPrintListsHierarchy` to output summation stream data.

Inputs:

- long ader_mat_stream_sum_ent
- FILE* fp
- int sum_level

Returns: void

A.9 ADER Testing

ADER includes an extensive suite of unit, integration, and system tests. The system tests are not supported by an automated testing environment. Rather, the system tests, found in the `System_Tests` directory of ADER, are composed of a README file describing the final results of the simulation who's input file is provided. To run a system test, execute the given input file with the specified command line options and compare the simulation output

with what the README file says to expect. For the integration and unit tests ADER should be compiled with the `-DADER_TEST` and `-DADER_INT_TEST` options and run with the simulation input found in the `inputs/Test_Input` directory titled "`test_input.txt`". A summary report titled "`TestResults.test`" will be produced in the executing directory. For a detailed accounting of the tests please see the source code. All tests are either, found in the `testcases.c` or have their own file with the name of "`TESTADER[NameOfTest].c`". The file `serp_tests.h` is an auxillery header file providing key information to the test management function, `runtests.c` which is responsible for executing the tests found in `testcases.c`. The standalone test functions, many of which are integration tests, are called from various places in the ADER framework, information which can be found in the function's doc-string. When compiled for unit and integration testing SERPENT2 will not work properly for any other purpose and must be recompiled for system testing or normal use.

A.10 ADER in Parallel

With regards to parallel computation there are two key facts: ADER is able to take advantage of shared-memory parallelization through the OpenMP interface, and ADER is unable to make use of distributed-memory parallelization which for SERPENT2 is hanlded through the MPI interface. There is no inherent reason ADER can not function inside of a distributed-memory environment - that capability simply does not yet exist.
Speaking to shared memory parallization, many functions which are part of the ADER suite are used in a threaded manner but are not labeled as thread-safe. This is because much of ADER's parallization happens on a per-cluster basis, each thread is given one cluster of materials to work with; that work being building or solving a matrix. Many of these threaded functions would fail to work as intended if they were not parallelized by the material cluster which they are operating on. Only those functions which are fully thread-safe, generally those which do not alter program memory, are labeled as such. There are two branching points for ADER threads. In `ADERCorrectTransportCycle` each material cluster is handed off to a thread which will then fill and solve the cluster composition optimization problem. Following the solution of these composition matrices a OpenMP barrier prevents any thread from moving forward until all threads have solved all their optimization problems afterwhich program flow is reduced back down to a single thread. In `ADERProcessMaterialAderData` each material is handed off to a thread which then proceeds to fill in that material's composition matrix information. Again, an OpenMP barrier prevents program continuation past this point until control is returned back to a single thread.

Appendix B

ADER User Manual

B.1 Preface

This document is intended for users of SERPENT2 wishing to utilize the **A**dvanced **D**epletion **E**Extension for **R**eprocessing (ADER) extension. **This manual assumes the reader is proficient with SERPENT2.** General information about SERPENT2 can be found at the SERPENT2 wiki:

serpent.vtt.ft/mediawiki/index.php/Main_Page

With regards to citing SERPENT2 and ADER, as stated on the SERPENT2 wiki general reference to SERPENT2 may be provided by - J. Leppanen, M. Pusa, T. Vittanen, V. Valtavirta, T. Kaltiaisenaho. *"The Serpent Monte Carlo code: Status, development, and applications in 2013"*. Ann. Nucl. Energy, **82** (2015) 142 - 150. Reference to ADER may be provided by - D. D. Wooten. *ADER - Advanced Depletion Extension for Reprocessing.* (2019).

ADER makes extensive use of the open-source library, Clp, part of the COIN-OR collection of packages. Supporting documentation and the source code for Clp can be found at:

https://github.com/coin-or/Clp

Of importance is that Clp is distributed under the Eclipse Public Licence which is not a copy-left licence but a rather forgiving open-source licence. Instructions for installing and linking the Clp libraries can be found in section B.12.
Examples of SERPENT2 input are given

```
in this font
```

SERPENT2 and ADER keywords, user inputs which must be a certain word, appear underlined as seen below. **Keywords not enclosed in brackets must appear in the position of the entry they are seen in. Keywords enclosed in brackets may appear anywhere after this opening keyword but must be followed by their input value if they have one.**

```
keyword
```

The following list of symbols are not to be considered "input". Rather, they are delimiters for this manual and should not be considered part of any ADER commands or inputs...

```
[ ] { } < > ( ) ; ...
```

Required user inputs are denoted...

```
[This_entry_is_set_by_the_user]
```

Optional user inputs are denoted...

```
<This_entry_may_be_excluded>
```

Required user choices from a list are denoted...

```
[{option_1; option_2}]
```

Optional user choices from a list are denoted...

```
<{option_1; option_2}>
```

User inputs which are required *if* a previous input was given are denoted...

```
<optional_input> (required_input_due_to_ealier_input)
```

If an example of a user input includes a structure such as...

```
[Entry_1] [some_number]
...
<Entry_n> (some_number)
```

This indicates that at least one instance of `Entry x` must be input by the user but as many instances of type `Entry x` may be input by the user. Additionally the presence of the second bracketed input indicates that each instance of `Entry x` requires an entry of `some_number`. Examples of generic SERPENT2 input, input templates, are given in a light-gray box with black mono-spaced font such as the example seen below...

```
mat [mat_name] [mat_den] <{vol,mass}> [mat_val]
<burn> (burn_segments)
[isotope_1_zai].[temp_lib] [iso_frac]
<isotope_n_zai>.(temp_lib) (iso_frac)
```

While any specific use examples, implementations of SERPENT2 input, are given in a black box with white mono-spaced font such as the example seen below...

```
mat water -1.0 vol 1.0 burn 0.0
1001.06c    2
...
1608.06c    1
```

B.2 Introduction

Please be sure to read the preface for important information.
In the most direct sense ADER seeks to accomplish two tasks: determining an optimal material composition given a set of constraints, and integrating the necessary composition adjustments into a nuclear material evolution model. ADER brings to Serpent 2 the ability to define groups of elements, isotopes, and chemicals; the ability to define relationships between these groups; and the ability to move these groups into, out-of, and between materials. Furthermore, ADER provides the ability to set k_{eff} targets, to prescribe mass transfers within a given system, and to set weighted oxidation state targets for materials. Bringing all of these capabilities together ADER employs the COIN-OR linear optimization (CLP) package to determine the material flows, as set by the user, that will best satisfy an optimization target, also set by the user. Sufficiently simplified - ADER is a material-composition, linear-optimization engine for the SERPENT2 burnup routines.

This manual serves as both a reference for the theory of ADER and as a guide to its usage. As such each section has a "Quick Reference" subsection where direct and simplified answers to common usage questions and input formatting can be found. However, many users will find that a deeper understanding of the theory behind ADER, as well as the nuances of its use, will serve them well. This manual is organized to be red through in a linear fashion by a first time user beginning with the concept of a "group" as it is used in ADER and building up the ADER toolkit from there.

To provide context for the explanation of the components of ADER components of an example simulation are built up throughout this manual. These components only serve as rough examples. That said, for this example consider a fuel salt for a liquid fuel molten salt reactor composed of LiF salt at 71.7 mol-fraction, BeF_2 salt at 16 mol-fraction, UF_3 at 0.023 mol-fraction, UF_4 at 2.277 mol-fraction, and ThF_4 at 10 mol-fraction. Take this salt to have an initial neutron multiplication factor of 1.0 (it does not in reality) and to be in an infinite lattice with a graphite moderator which takes up 50% of the volume. The SERPENT2 material based off this example is shown below. **The keyword "ader" must be included in the definition line of any material which is to be managed by the ADER extensions to SERPENT2 - i.e. the material is either connected to a stream or to a conditions block.** ADER input is entered as a "regular" component of SERPENT2 input files.

```
mat FuelSalt -2.805 vol 1 burn 0 ader
3006.06c     0.00028
3007.06c     0.28323
4009.06c     0.06328
9019.06c     0.60450
90232.06c    0.03954
92233.06c    0.00046
92238.06c    0.00864
```

B.3 Groups

At the most fundamental level a group in ADER is a list of proportions. This list describes the relative proportions of elements within the group. These elements may or may not have specified isotopic proportions and elements without a specified isotopic composition are permitted in the same group as elements *with* a specified isotopic composition. Consider a group which specifies that fluorine be four times as abundant as uranium, within the group. This group could be used to describe a single molecule of UF_4 or, just as equivalently, one mol of uranium and four mols of fluorine all in a bucket on a lab bench. A group is not defined by a quantity of material, but rather a recipe for a material. Just like defining materials in SERPENT2, the proportions of an element within a group are normalized to unity and the proportions of an isotope within an element are normalized to unity. As such groups in ADER are defined according to four methods A, B, C, or D as seen bellow.

Method A:

```
grp [Name]
[Element_1] [Element_1_frac]
...
<Element_n> (Element_n_frac)
```

Method B:

```
grp [Name]
[Element_1] [Element_1_frac] <isos> (n_isos)
(Iso_1) (Iso_1_frac)
<Iso_n> (Iso_n_frac)
...
<Element_x> [Element_x_frac] <isos> (b_isos)
(Iso_1) (Iso_1_frac)
<Iso_b> (Iso_b_frac)
```

Method C:

```
grp [Name]
[Element_1] [Element_1_frac]
...
<Element_x> [Element_x_frac] <isos> (b_isos)
(Iso_1) (Iso_1_frac)
<Iso_b> (Iso_b_frac)
```

Method D:

```
grp [Name] [sum]
[Name_of_group_1] 1
...
<Name_of_group_n> 1
```

The keyword `grp` begins the group section input. All input following this word, until the next keyword is found, is taken as input for the aforementioned `grp`. The `Name` entry is an identifier by which the group will be located throughout the code - as such all groups require a unique name. In method A a group is defined only by the proportions of its elements. The relative abundance of each element within the group is denoted by `Element_x_frac` while the values for `Element_x` must be the alphabetic periodic table shorthand for that element, Ag for gold as an example. Elements with no prescribed isotopic composition are permitted to take on any combination of their own isotopes.

In method B a group is defined in which each element has a list of specified isotopic proportions. The values for `Iso_n` must be the alphanumeric name of the isotope in the form "Alpha-A", or U-233 for ^{233}U The proportions entered, `Iso_n_frac`, are of the isotope relative to the element as a whole. In method C a group is defined in which some of its elements have a specified list of isotopic proportions and some do not. Those elements which do not have a prescribed list of isotopic proportions are permitted to have any combination of their own isotopes. Lastly, in method D a new type of group is introduced - the summation group. A summation group is defined as the sum of the groups named in its list of component groups. These component groups may be defined using methods A, B, C, or D, allowing for nested summation groups. The value which follows a group name in a summation group definition may be any value greater than 0; however, any value other than 1 could lead to a non-physical answer in the simulation. Please see section B.11 for a more detailed explanation.

The `Element_x_frac` values for each group are all summed together and then normalized to this sum. This same procedure is done for all isotopic values within an element's list. For example, consider the group `gUF4` defined below, it is composed of 20% uranium and 80% fluorine. The fluorine is permitted to have any combination of fluorine isotopes while the uranium in this group is specified to be 5% ^{233}U and 95% ^{238}U.This is an example of method C of group definition.

```
grp gUF4
U   1   isos    2
U-233   5
U-238   95
F   4
```

The group `gFLiBe` is defined below as an example of method A, group `gUF3` is defined below as an example of method B, while group `gUF` is defined below as an example of method D. These groups will be used throughout this manual in examples.

```
grp gFLiBe
Li  71.7
Be  16
F   103.7

grp gUF3
F   3   isos    1
```

```
F-19     1
U   1    isos     1
U-233    1

grp gUF sum
gUF4     1
gUF3     1
```

The following paragraphs assume some level of familiarity with ADER and will likely be unclear until sections B.4 and B.5 have been reviewed.

Groups are used to define four structures in ADER; ranges, ratios, streams, and summation groups. These structures are covered in later sections of this manual but their interaction with groups requires a comment here. When a group is used in a range or ratio structure, or is used in a summation group which is part of a range or ratio structure, and this structure is attached to a material by a conditions block, the group in question is added to the material's list of possible recipes. These recipes are the options into which the material's constituent isotopes may be sorted during the optimization step, discussed in section **??**. Range and ratio structures, in the same material, make use of the same group list. That is to say, in the following example seen below, in which the conditions block `example_block` is attached to the material `FuelSalt`, the group `gFLiBe` must **both** be between 20 and 80% of `FuelSalt`'s atomic density **and** be four times as abundant as the group `gUF4`. There are **not** now two copies of the group `gFLiBe` assigned to the material `FuelSalt`. The `rng` and `rto` structures each refer to the **same** group within a single material, in this instance the material `FuelSalt`.

```
conditions example_block
rng gFLiBe min 0.2 max 0.8
rto gFLiBe val 4 grp2 gUF4

mat FuelSalt -2.805 vol 1 burn 0 ader cnd example_block
3006.06c     0.00028
...
```

The groups used to define group-class streams are used solely as recipes. The groups used to define group-class streams *only* specify the relative proportions of the mass carried by the stream in question. For a stream to be connected to a material, that material *does not* need to have that group in its list of possible recipes. For instance, `FuelSalt` to which the conditions block above, `example_block`, is attached could be the sink for a group-class stream defined using the group `gLi`, a group of elemental lithium, even though there is no usage of the group `gLi` in the conditions block `example_block`. A group-class stream's usage of a group only stipulates what proportions the mass that stream carries must have. If the mass the stream carries originates from a material the isotopes which compose the stream's load may come from any group or no group at all inside of that material. If the mass the stream carries ends in a material the isotopes entering the material may go into any group

into which they fit the recipe, or no group at all if the isotope is not a "controlled" isotope - a topic discussed in section B.4.3.

Again, the groups defined using the `grp` keyword are only recipes. A given recipe can be used to define multiple structures. Streams all create their own copy whereas range and ratio restrictions in the same material refer to the same group in a material's recipe list. As an example, in the input below, there exist four digital versions of the group `gFLiBe`, the original `grp` recipe, the group which has been added to the material `FuelSalt`'s recipe list, and each group created for each stream; and while these two streams represent two distinct stream objects, the optimization line seen in the conditions block will minimize the *sum* of the two streams because they have each been defined with a group of the same name.

```
grp gFLiBe
Li  71.7
Be  16
F   103.7

conditions example_block
rng gFLiBe min 0.2 max 0.8
rto gFLiBe val 4 grp2 gUF4
opt dir min type spec_stream gFLiBe

mat FuelSalt -2.805 vol 1 burn 0 ader cnd example_block
3006.06c    0.00028
...

stream to FuelSalt type feed form cont group gFLiBe
stream to FuelSalt type reac form disc group gFLiBe
```

B.3.1 Quick Reference

- The proportions of elements within a group are normalized to unity.
- The proportions of isotopes within an element are normalized to unity.
- Only isotopes belonging to the same element may be listed as a component isotope of that element.
- General Input Structure where `Element_x` must be the alphabetic periodic table designation for the element, and `iso_x` must be the alphabetic periodic table designation for the parent element followed by a dash and then the atomic number of the isotope in question. The value which follows a group name in method D of defining a group may be any value greater than 0; however, any value other than 1 could generate a non-physical answer for a given simulation.

```
grp [Name] <sum>
[{Element_1; group_1}] [{Element_1_frac; 1}] ...
<isos> (num_isos)
(iso_1) (iso_1_frac)
<iso_n> (iso_n_frac) ...
<{Element_n; group_n}> [{Element_n_frac; 1}] ...
<isos> (num_isos)
(iso_1) (iso_1_frac) <iso_n> (iso_n_frac) ...
```

B.4 Conditions Blocks

A conditions block is the structure through which limits on a SERPENT2 material are defined. A conditions block consists of, minimally, the elements seen below where `conditions` is the keyword and `Name` is the unique identifier given to this specific conditions block.

```
conditions [Name]
```

Consider this conditions block `limits_block` as defined below.

```
conditions limits_block
```

A conditions block, and all of the limitations it may carry, are applied to SERPENT2 materials by including the key and value pair, `cnd [Name]`, in the material's definition; the information that follows the `mat` keyword.

```
... [cnd] [Name] ...
```

Building upon the example presented in the introduction, to attach a conditions block with the name of `limits_block` the material definition line would be amended to look like...

```
mat -2.805 FuelSalt vol 1 burn 0 ader cnd limits_block
```

The same conditions block can be attached to multiple materials with each material receiving its own unique set of the limitations - this reduces input duplication. To add limitations to a conditions block there exist six possibilities: ranges, ratios, control tables, oxidation tables, preservation options, and optimization options. Each will be covered below in the following subsections.

B.4.1 Ranges

Using the `rng` keyword the fraction of a material, by percent atom per cubic centimeter ($\frac{\%}{cm^3}$) which must adhere to the recipe of a specific group may be set. This fraction may be either a single value, seen in method A, or a range of values, as seen in method B. Values less than 0 are not accepted but there is no upper limit for the material fractions. Using a group in a `rng` structure which is attached to a material via a condition block is one of two methods to "attach" a group to a material - the meaning of this is expanded upon in section B.4.3

Method A:

```
rng [grp_name] val [value]
```

Method B:

```
rng [grp_name] [min] [min_value] [max] [max_value]
```

Take g_u to be the fraction of a material's atomic density which is attributed to group u, in which case method A sets up the relation seen in equation B.1 and method B sets up the relation seen in equation B.2.

$$g_u = \texttt{[value]} \tag{B.1}$$

$$\texttt{[min\_value]} \leq g_u \leq \texttt{[max\_value]} \tag{B.2}$$

Now consider the conditions block shown below which was attached to the material `FuelSalt` at the beginning of this section, B.4.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
```

By adding a `rng` input to the conditions block `limits_block` material `FuelSalt` is now required to have at least 3% and no more than 10% of its atomic density accounted for by isotopes which would fit the recipe of group `gUF4`. Furthermore, these isotopes may not be counted towards inclusion in any other group structure attached to the material `FuelSalt`.

B.4.2 Ratios

The keyword `rto` is used to describe the permitted relative abundance of groups within a material. Like `rng` structures `rto` entries can be make according to methods A or B seen below. Values equal to or less than 0 are not accepted. Using a group in a `rng` structure which is attached to a material via a condition block is one of two methods to "attach" a group to a material - the meaning of this is expanded upon in section B.4.3
Method A:

```
rto [grp1_name] [val] [value] [grp2] [grp_2_name]
```

Method B:

```
rto [grp1_name] [min] [min_value] [max] [max_value]
[grp2] [grp_2_name]
```

Method A sets up the relation seen in equation B.3 and method B sets up the relation seen in equation B.4.

$$\frac{g_1}{g_2} = \texttt{[value]} \tag{B.3}$$

$$\texttt{[min\_value]} \leq \frac{g_1}{g_2} \leq \texttt{[max\_value]} \tag{B.4}$$

Now, consider adding the following line to the conditions block, `limits_block`, which has served as the example throughout this section.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
rto gFLiBe  min 4.0 max 99  grp2    gUF
```

This `rto` line specifies that in the material `FuelSalt` the fraction of the material's atomic density which is accounted for by the group `gFLiBe` must be no less than 4 times as prevalent and no more than 99 times as prevalent as the fraction of the material's atomic density which is accounted for by the group `gUF` - which is a summation group.

B.4.3 Control Tables

In the ADER framework there is no requirement that all of a material's atomic density be accounted for by group structures - groups are permitted to occupy as little or as much of a material's composition as the optimal solution requires. Consider a group, `gNi`, which is defined as elemental nickel and which is limited to be less than 2% of a material's atomic density. One way to meet this requirement is simply to assign none of the nickel isotopes in the material to the group `gNi`. In that case 0% of the material's atomic density is accounted for by the group `gNi`.

Control tables offer a means to prevent the trivial answer reached above. A control table is a list of elements and isotopes. When attached to a material a control table specifies that any isotopes in the material which are listed on the control table, or which are a member of an element listed on the control table, must be fully accounted for by a group structure attached to the material. Groups become attached to a material by being a component of an `rng` or `rto` entry in a conditions block which is assigned to the material in question. In the example above, if there are no other groups for the nickel isotopes to go into, if nickel is a controlled element in the material in question, all of the nickel isotopes in the material would be forced into the `gNi` group. An element or isotope not listed on a control table is referred to as a "free" element or isotope as these constituents are not bound to be in groups.

A control table is defined in the following manner where elements are denoted by their alphabetic periodic table designation and isotopes by adding a dash followed by the atomic number of the isotope in question to the alphabetic periodic table designation for the parent element: i.e. `U` for uranium and `U-233` for ^{233}U.

```
control [table_name]
[element_or_isotope]
...
<additional_element_or_isotope>
```

For example, consider a control table which specifies that all lithium, beryllium, ^{233}U, and thorium must be accounted for by group structures, as seen below.

```
control c_table
Li Be U-233 Th
```

A control table is attached to a conditions block via the entry...

```
[cnt] [table_name]
```

Attaching the control table, `c_table`, from above to the conditions block used in this section is seen below.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
rto gFLiBe  min 4.0 max 99  grp2    gUF
cnt c_table
```

B.4.4 Oxidation Tables

Oxidation tables are a means of setting a limitation on a weighted sum which is done over all the *elements* in a material. Oxidation tables are constructed as seen below where `value` is multiplied by `weight_value`, if applicable, which produces the weight in the weighted sum done over the elements which are enumerated in the oxidation table. Elements are entered by their alphabetic periodic table designation. Any real number is a valid input for both `value` and `weight_value`.

```
oxidation [Name]
[first_element] [value] <weight> (weight_value)
<nth_element> (value) <weight> (weight_value)
```

As an example consider the oxidation table `oxi_table` defined below. Any elements excluded from an oxidation table's list are given a default value of 0.

```
oxidation oxi_table
H   1
O   -2
```

Oxidation tables are attached to conditions blocks either by method A or method B. Method A:

```
oxi [Name] val [target_val]
```

Method B:

```
oxi [Name] [min] [min_val] [max] [max_val]
```

Taking ρ_e to be the fraction of a material's atomic density which is attributable to element e an oxidation table, in conjunction with its implementation in a conditions block attached

to said material, establishes the relationships seen in equations B.5 and B.6 corresponding to methods A and B respectively.

$$\sum_{e}^{E} \rho_e v_e w_e = \texttt{target\_val} \tag{B.5}$$

$$\texttt{min\_val} \leq \sum_{e}^{E} \rho_e v_e w_e \leq \texttt{max\_val} \tag{B.6}$$

Attaching the oxidation table `oxi_table` to the conditions block used throughout this section is demonstrated below using method B.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
rto gFLiBe  min 4.0 max 99  grp2    gUF
cnt c_table
oxi oxi_table min -0.02 max 0.06
```

B.4.5 Preservation

Inclusion of the following keyword pair...

```
pres mols
```

into a conditions block adds the limitation to the material that any influx of matter must be equally balanced by a removal of matter with an equal number of atoms; i.e. the atomic density of a material is not permitted to change due to mass flows - only nuclear depletion effects. Only mass flows managed by group-class streams, covered in section B.5, are included in this balance. The majority of simulations will find this option to be necessary; otherwise unintended behavior may result.

Including this modifier into the conditions block used throughout this section is seen below.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
rto gFLiBe  min 4.0 max 99  grp2    gUF
cnt c_table
oxi oxi_table min -0.02 max 0.06
pres mols
```

B.4.6 Optimization

As ADER is a linear optimization suite, there must be an optimization target. Every material cluster, a concept covered in section B.5, with at least one group-class stream, must have one

and only one optimization target. Optimization targets are attached to material clusters via a conditions block which is attached to a member of a material cluster. There are nine methods to set optimization targets. The direction of optimization, maximization or minimization, is set in the optimization entry of the conditions table as well. Many of the optimization targets involve concepts related to streams which are covered in section B.5.
Method A sets the optimization target as the total value of all feed/reac/redox/remv type streams:

```
opt [dir] [{min; max}] [type]
[action] {[feed; reac; redox; remv]}
```

Method B sets the optimization target as the total value of all feed and remv type streams:

```
opt [dir] [{min; max}] [type] [action] [feed_and_remv]
```

Method C sets the optimization target as the total value of all streams:

```
opt [dir] [{min; max}] [type] [action] [streams]
```

Method D sets the optimization target as the total value of all streams which have a valid SERPENT2 material for both the source and sink:

```
opt [dir] [{min; max}] [type] [action] [transfers]
```

Method E sets the optimization target as the total fraction of a material's atomic density which is attributed to the named group. This group *must* have been assigned to a material in the material cluster via either a `rng` or `rto` conditions block entry.

```
opt [dir] [{min; max}] [type] [group] [group_name]
```

Method F sets the optimization target as the total value of all group-class streams defined using the group of name `group_name`.

```
opt [dir] [{min; max}] [type] [spec_stream] [group_name]
```

As an example the optimization target of minimizing all feed type streams is added to the conditions block used throughout this section.

```
conditions limits_block
rng gUF4    min 0.03    max 0.10
rto gFLiBe  min 4.0 max 99  grp2    gUF
cnt c_table
oxi oxi_table min -0.02 max 0.06
pres mols
opt dir min type action feed
```

B.4.7 Quick Reference

- Conditions blocks are defined as follows...

```
conditions [Name]
```

- Conditions blocks are attached to materials using the `cnd [Name]` key and value pair in a material's definition - i.e. `mat my_mat cnd my_conditions_block`
- Ranges are defined inside a conditions block as follows...

```
rng [grp_name] [{val; min/max}] [value] (max/min) (value)
```

- Ratios are defined inside a conditions block as follows...

```
rto [grp1_name] [{val; min/max}] [value] (max/min) (value)
    grp2 [grp2_name]
```

- Control tables are defined as follows...

```
control [table_name]
[element_or_isotope]
...
<element_or_isotope>
```

- Control tables are added to conditions blocks as follows...

```
cnt [table_name]
```

- Oxidation tables are defined as follows...

```
oxidation [table_name]
[element_x] [value] <weight> (weight_value)
[element_n] [value] <weight> (weight_value)
```

- Oxidation tables are added to conditions blocks as follows...

```
oxi [table_name] [{val; min/max}] [value] (max/min) (value)
```

- To turn on atom-density conservation for a material, add the following line to a conditions block attached to said material...

```
pres mols
```

- Every material cluster with one or more group-class streams must have one and only one `opt` entry attached to one of the conditions blocks which is itself attached to one of the materials in a cluster. `opt` entries are added to conditions blocks as follows...

```
opt [dir] [{min; max}] [type] [{action; group; spec_stream}]
    [{feed; feed_and_remv; reac; redox; remv; streams;
        transfers; group_name}]
```

B.5 Streams

Streams are the structures by which ADER moves mass into, out of, and between SERPENT2 materials. Streams are defined as seen below...

```
stream <to> (name_of_sink_material)
<from> (name_of_source_material)
    [type] [{feed; reac; redox; remv}] [{group; rem}] [name]
    [form] [{cont; disc; prop}] (frac) (frac_value)
```

The `to` and `from` entries specify the stream's source and sink for the mass which the stream moves. Sources and sinks which are given must be valid SERPENT2 materials. Streams do not require *both* a source and a sink, but rather, at least *one* of either a source or sink. In the case of a missing source the mass which is brought into the sink material is assumed to come from an infinite, non-reactive (chemically and neutronically) supply existing outside the simulation bounds. In the case of a missing sink the mass which is removed from the source material is assumed to disappear - it leaves the simulation boundary.

Streams which have elements with unspecified isotopic compositions take these element's isotopic compositions to be equal to that of the elements in the source material at the beginning of the burnup step. These proportions are updated on each ADER criticality search to account for the changes possibly induced by discrete form streams. Streams which have elements with unspecified isotopic compositions and a valid SERPENT2 material as a sink must have a valid SEPRENT2 material as a source.

Material clusters are collections of SERPENT2 materials which are connected via streams. If a stream connects two materials, having a valid SERPENT2 material for each source and sink, these two materials become part of the same material cluster. Because materials may have streams coming from and going to many other materials, not all materials in a material cluster will be directly connected by streams; rather, some materials in a material cluster may happen to only be connected to one another through a series of intermediate streams and materials. Materials in the same cluster share an optimization solution, that is, their compositions are optimized collectively according to the singular optimization target provided, discussed in section B.4.6. Of less impact to the user, material clusters also share a collective burnup solution though each material may have its own flux and cross sections. Materials in the same cluster must have the same initial isotopes listed in their definitions whether or not these isotopes exist in each material at the beginning of the simulation. If an isotope needs to be listed in a material for which its concentration is zero, simply include the isotope in the material's definition with a zero value next to its ZAI identifier.

The `type` entry has no effect on stream behavior or implementation. Rather, it serves only as a tag for the `opt` entries (discussed in section B.4.6) and as a possible organizational tool for the user.

The following choice of keywords, `group` and `rem`, highlights the most important distinction between streams; the difference between group-class (`group`) and table-class (`rem`) streams. While the details of this distinction will be covered in depth in the following sub-

sections it is sufficient to say, for now, that group-class streams move mass which follows the recipe outlined by a group while table-class streams move mass of which the proportions are determined using a table, a `removal` table (covered in section B.5.2). Last but not least, an additional distinction is that group-class streams are variables, the amount of mass which they move is determined by solving the material optimization problem, discussed in section **??** while table-class streams move a fixed amount of mass according to user input. Group-class streams have the name of a group following the keyword `group` while table-class streams have the name of a removal table following the keyword `rem`. **Table-class streams can be used to feed mass into a material. The use of the keyword `removal` is a legacy-term and does not explicitly mean ‘‘removal” in the sense of the English word**.

The `form` keyword and its associated values, `cont`, `disc` and `prop`, designate how a stream behaves in time. The optimization solution, as discussed in section **??**, determines the total mass transfers from the group-class streams interfacing with a material cluster needed to bring the materials in the material cluster to their optimal composition. How this mass is moved over time is up to the user through the use of the `form` keyword. Streams designated as `cont` are “continuous” streams. These streams move mass throughout a burnup step at a constant rate per second. Streams designated as `disc` are “discrete” streams. These streams move mass as an instantaneous input happening before a burnup step begins. Streams designated as `prop` type streams are “proportional” streams and their method of mass transfer is discussed in greater detail in the following sections, B.5.1.3 and B.5.2.3.

The `frac` keyword and its associated value are discussed in sections B.5.1 and B.5.2.

An important note, mentioned in several spots throughout this manual, is that only “disc” form streams impact ADER’s reactivity iteration scheme. If ADER is instructed to iterate on the material composition to match user specified reactivity targets, ADER will only apply the actions of discrete form streams on each iteration. Any continuous and proportional streams will not have their effects on material reactivity incorporated until after the current burnup step is complete. Additionally, any given material constraint may be out of bounds during some point in a burnup step if streams of mixed type are used together as the optimal conditions for the material rely on the full impact of all streams. As a closing reminder, ADER *does not* account for nuclear burnup in the *current* burnup step. The effects of nuclear burnup on composition are assessed by ADER at the beginning of the next burnup step - i.e. ADER does not predict the effects of nuclear burnup.

B.5.1 Group-class streams

Group-class streams, those streams using the keyword `group`, operate differently from table-class streams. Group-class streams are options, given by the user to ADER, for moving mass. As discussed in section B.11 the amount of mass moved by group-class streams is determined by the optimization solution. That is, the content of group-class streams is determined so that the optimal material composition for the material cluster may be realized.

The mass which a group-class stream moves is described by the group named after the `group` keyword. This named group describes the proportions the mass the stream carries must have. The amount of mass is determined by the optimization solution. The optimization solution operates on a volume normalized basis. As such, take the initial units of any solution for a stream value to be[1] $\frac{\texttt{atoms}}{cm^3}$ where "`atom`" represents a unit of the group-class stream. This unit is composed of fractional isotopes in accordance with the recipe of the group-class stream. Take this value to be denoted $s_{n:x}$ where n denotes different streams and x takes on either c, d, or p representing continuous, discrete, and proportional form streams.

Group-class streams do not make use of the `frac` keyword and as such their input template appears as follows...

```
stream <to> (name_of_sink_material)
<from> (name_of_source_material)
    [type] [{feed; reac; redox; remv}] [group] [name]
    [form] [{cont; disc; prop}]
```

B.5.1.1 Continuous group-class streams

Continuous group-class streams move an equal amount of mass per unit time over the length of a burnup step. Taking Δt to be the total number of seconds in the current burnup step continuous group-class streams deliver or remove $c_n \frac{\texttt{atoms}}{cm^3 s}$ to their sink or source material's respectively where c_n is defined in equation B.7.

$$c_n = \frac{s_{n:c}}{\Delta t} \tag{B.7}$$

As an example consider below the continuous group-class stream formed using the `gFLiBe` group from section B.3 for which there is no source and for which the sink is the material `FuelSalt`.

```
stream to FuelSalt group gFLiBe form cont type feed
```

B.5.1.2 Discrete group-class streams

Discrete group-class streams move their entire mass load, $s_{n:d}$, collectively and instantaneously following the optimization solution and before the transport sweep preceding the burnup calculation. With further detail available in section B.11 discrete group-class streams have a unique aspect different from all other streams. Every burnup step ADER iterates over the optimization solution and a transport sweep until the system analog neutron multiplication factor, k_{eff}^{analog}, is within user defined bounds. Every time an optimization solution is arrived at, if there are discrete streams, these streams will have some value $s_{k,n:d}$ where k denotes the current iteration over the optimization solution for the current burnup step. The

[1] Developers note: The actual units of stream return values in the code are $\frac{\texttt{atoms}}{cm \cdot barn}$

changes in material composition induced by these discrete stream flows, $s_{k,n:d}$, are applied before the next transport sweep meaning that the system compositions reflect the actions of discrete group-class streams. Should another solution of the optimization problem, for the same burnup step, be required, the changes induced by these discrete group-class stream flows, $s_{k,n:d}$, are not undone. Rather, the final $s_{n:d}$ value reported for discrete group-class streams per burnup step is given by equation B.8.

$$s_{n:d} = \sum_{k}^{K} s_{k,n:d} \tag{B.8}$$

As an example of a discrete group-class stream consider the stream below formed using the group `gUF4`, again, with no source but the material `FuelSalt` as the sink.

```
stream to FuelSalt group gUF4 form disc type reac
```

B.5.1.3 Proportional group-class streams

Proportional group-class streams attempt to move mass in proportion to the mass already present. Proportional group-class streams approximate a decay-like proportional constant, $\lambda_{i,n:p}$, which in a system with no other effects acting on isotope concentration, would lead to the desired change in the isotope's, i, concentration. This is almost never the case in an ADER simulation and as such the realized change in an isotope's concentration will likely be different from the desired amount of change. As such, the values reported in the output by ADER for all proportional streams, group-class or table-class, are calculated exactly during the burnup solution such that they reflect the actual amount of the isotope moved, not the pre-calculated amount which is likely different. Regardless, considering the role that group-class streams tend to play in ADER simulations the usage of proportional group-class streams should be undertaken with great caution and a thorough understanding of how the simulation parameters will interact to affect the proportional group-class stream constants. Proportional group-class streams may not be formed using summation groups.

For a proportional stream removing a quantity, $s_{n:p}$, from a material, m, the isotopic proportional constants, $\lambda_{i,n:g}$ for isotope i as modified by group stream n are calculated, before the burnup step and *after* the application of any discrete stream effects (such that the proportional constants are calculated from the updated material composition values), as seen below in equation B.9 where $f_{i,n:p}$ is the proportion of the stream n accounted for by isotope i, $\rho_{m,k=0}$ is the source material atomic density at the beginning of the zeroth iteration before any discrete stream actions have been applied to any materials, N_i is the current atomic density of isotope i, and Δt is the total time of the current burnup step. In the case of a desired 100% removal of an isotope ADER approximates the term $(s_{n:p} f_{i,n:p} \rho_{m,k=0} N_i^{-1})$ as $(1 - 10^{-8})$.

$$\lambda_{i,n:g} = ln(1 - s_{n:p} f_{i,n:p} \rho_{m,k=0} N_i^{-1})(\Delta t)^{-1} \tag{B.9}$$

For a proportional stream adding a quantity of mass to a material the isotopic proportional constants are calculated, before the burnup step and *after* the application of any discrete stream effects (such that the proportional constants are calculated from the updated material composition values), as seen below in equation B.10.

$$\lambda_{i,n:g} = ln(1 + s_{n:p} f_{i,n:p} \rho_{m,k=0} N_i^{-1})(\Delta t)^{-1} \tag{B.10}$$

During the construction of the burnup matrix, discussed in section B.11.7 these proportional constants are added to the respective decay constants of the isotopes they seek to modify.

As an example of a proportional group-class stream consider the stream below formed using the `gUF4` group and using material `FuelSalt` as a source, having no sink.

```
stream from FuelSalt group gUF4 form prop type remv
```

B.5.2 Table-class streams

Table-class streams, unlike group class streams, are directions from the user to ADER to move a specific quantity of mass in a specific manner. A stream is classified as a table-class stream when the keyword `rem` is used in the stream definition. This keyword is always followed by the name of a `removal` table. **The use of the word "removal" for the name of this structure is an artifact from earlier code development. These tables can be used to move mass into, out of, and between SERPENT2 materials. The materials listed as their `to` and `from` entries, sink and source respectively, operate as sink and source respectively. The name of this structure `removal` has no relation to the English word "removal" other than spelling.** Removal tables are defined elsewhere in the code using the format below where elements are entered using their alphabetic periodic table designation, so `U` for uranium, while isotopes are entered as elements followed by `-A` where A is the atomic number of the isotope; `U-235` for ^{235}U.

```
removal [Name]
[element_or_isotope_1] [table_value]
...
<element_or_isotope_n> (table_value)
```

Both elements and isotopes may be entered in the same removal table. If a `table_value` ,q, is entered for both an element and one or more of its constituent isotopes, the isotopes for which a `table_value` was entered will retain their corresponding value rather than be overwritten by their parent element's value - i.e. an elemental `table_value` is a default for all child isotopes of that parent element but this default is overwritten by any entry for a child isotope. All q values must be greater than or equal to zero.

As an example, consider the removal table seen below. In this removal table, `rem_table`, all isotopes, i, of uranium have an associated q_i value of 0.1, except for ^{233}U which has a q_i value of zero.

```
removal rem_table
U      0.1
U-233  0.0
```

A single removal table may be used to define multiple streams, each of which will implement the removal table in its own designated fashion.

All table-class streams require the use of the keyword `frac` and the input of its associated value which must be equal to or greater than zero. As such the input template for table-class streams appears as such...

```
stream <to> (name_of_sink_material)
<from> (name_of_source_material)
    [type] [{feed; reac; redox; remv}] [rem] [name]
    [form] [{cont; disc; prop}] [frac] [frac_value]
```

In the following subsections many equations and parameters will depend on a material value such as density or volume. All material values in reference to table-class streams are taken at the source material if there is one. If there is no source material these material values are taken at the sink material.

B.5.2.1 Continuous table-class streams

Continuous table-class streams move $s_{n:c}$ amount of material every burnup step where $s_{n:c}$ is defined by equation B.11 where $q_{i,n:c}$ is the `table_value` given for isotope i by the removal table used to create stream n, N_i is the number density for that isotope (taken after the effects of discrete streams have been applied), r_n is the `frac_value` given for stream n, and V_m is the material volume.

$$s_{n:c} = V_m \sum_{i}^{I} q_{i,n:c} r_n N_i \tag{B.11}$$

From equation B.11 it is clear that the product of a value from a removal table and the value given for the stream's `frac` entry is the percentage, by atomic density, of that removal table entry to move during each burnup step. Continuous table-class streams move this mass evenly over time. The transfer rate, $h_i \frac{\texttt{atoms}}{s^{-1}}$ for isotope i is given by equation B.12 where Δt is the length of the current burnup step in seconds.

$$h_i = V_m q_{i,n:c} r_n N_i \Delta t^{-1} \tag{B.12}$$

It is important to remember that isotopes derive their $q_{i,n}$ values from their parent element's q_e value given in the removal table used to define stream n unless the isotope in question was given its own q_i value in the same removal table.

As an example of a continuous table-class stream consider the following...

```
stream from FuelSalt rem rem_table type redox form cont fr
```

B.5.2.2 Discrete table-class streams

Discrete table-class streams move $s_{n:d}$ amount of material every burnup step where $s_{n:d}$ is defined by equation B.13 where $q_{i,n:d}$ is the `table_value` given for isotope i by the removal table used to create stream n, N_i is the number density for that isotope (taken before the effects of discrete streams have been applied), and r_n is the `frac_value` given for stream n.

$$s_{n:d} = V_m \sum_{i}^{I} q_{i,n:d} r_n N_i \tag{B.13}$$

From equation B.11 it is clear that the product of a value from a removal table and the value given for the stream's `frac` entry is the percentage of that removal table entry to move during each burnup step. Discrete table-class streams move this mass instantaneously at the beginning of each burnup step. The transfer amount, h_i for isotope i is given by equation B.14 where Δt is the length of the current burn up step in seconds.

$$h_i = V_m q_{i,n:c} r_n N_i \Delta t^{-1} \tag{B.14}$$

It is important to remember that isotopes derive their $q_{i,n}$ values from their parent element's q_e value given in the removal table used to define stream n unless the isotope in question was given its own q_i value in the same removal table. With regards to discrete stream behavior throughout ADER criticality iterations, the effects of discrete table-class streams are applied on the zeroth iteration for each burnup step, and no more than that.

As an example of a discrete table-class stream consider the following...

```
stream to FuelSalt rem rem_table type feed form disc frac
```

B.5.2.3 Proportional table-class streams

Proportional table-class streams *do not* move a specific quantity of material in a burnup step. Rather, proportional table-class streams modify the decay constant of the isotopes in question - this modification may be slight or may even be of a magnitude to turn a decay constant into a production constant. The stream-constants, $\lambda_{i,n:t} \frac{1}{s^{-1}}$ for isotope i as modified by table-class stream n are determined according to equation B.15.

$$\lambda_{i,n:t} = q_{i,n:p} r_n \tag{B.15}$$

Concerning the inclusion of $\lambda_{i,n:t}$ into the burnup matrix, while this is covered in detail in section B.11.7, it is sufficient to say for now that $\lambda_{i,n:t}$ is added to the isotope's decay constant if there is no source material, this isotope being in the sink material. If there is a source material $\lambda_{i,n:t}$ is subtracted from the source isotope's decay constant and this production term is added to the isotope in the sink material such that the transfer rate is dependent on the source material isotopic concentration.

As an example of a proportional table-class stream consider the following...

```
stream from FuelSalt rem rem_table type remv form prop fra
```

B.5.3 Quick Reference

- Streams are defined as follows...

```
stream <to> (name_of_sink_material)
<from> (name_of_source_material)
    [type] [{feed; reac; redox; remv}] [{group; rem}] [name]
    [form] [{cont; disc; prop}] (frac) (frac_value)
```

- Removal tables are defined as follows...

```
removal [Name]
[element_or_isotope_1] [table_value]
...
<element_or_isotope_n> (table_value)
```

- Streams require a source or a sink. They may have both, but both are not required.
- Streams with elements lacking an isotopic composition derive the element's isotopic composition from the source material.
- Streams with elements lacking an isotopic composition AND with a valid SERPENT2 material as a sink MUST have a valid SERPENT2 material as a source.
- Material clusters are collections of materials which are connected by ADER streams
- Group-class streams change their delivered mass each burnup step according to the optimization solution.
- Table-class streams move user-defined quantities of mass each burnup step
- Proportional group-class streams may not be formed using summation groups.

B.6 Criticality Control

In many nuclear burnup simulations the overall neutron multiplication factor of the system in question has some desired value or range it should hold. ADER provides the user the ability to set overall system k_{eff}^{analog} targets, a minimum and a maximum value, set using the below input. The default values for `kmin` and `kmax` are zero and ∞ respectively.

```
kmin [minimum_k_value]
kmax [maximum_k_value]
```

For instance, the below input would place a desired lower bound on k_{eff}^{analog} of 1.0 and an upper bound of 1.001.

```
kmin 1.0
kmax 1.001
```

As discussed in section B.11, ADER only considers the neutron multiplication factor of a single SERPENT2 material. This material must be designated by the user using the keyword and value pair, `rhow 1.0`, in the material's definition line before the list of its constituent isotopes such as seen below...

```
mat FuelSalt -2.805 vol 1 burn 0 rhow 1.0 ader
```

Also discussed in section B.11 is the "probability of absorption" factor used as an approximation of the effects of the rest of the system outside of the singular material considered for criticality control. While a detailed discussion of this approach is left to section B.11 for now it will be said that this method of assessing criticality in systems is expected to produce reasonable and well-behaved results for systems in which there is a single material which is both largely responsible for nuclear criticality and weakly coupled neutronically to its surroundings.

ADER has a limited iteration scheme for criticality search. This scheme is visually depicted as a flowchart in figure **??**. At the beginning of each burnup step ADER builds and solves the optimization problem for all material clusters. Following this the determined actions for discrete type streams, both group-class and table-class, are applied to change material compositions. Following this a new Monte Carlo transport sweep is executed with all the same population and cycle parameters as the transport sweep for a burnup step. Following this transport sweep if the system k_{eff}^{analog} is within the user set bounds, or no iterations remain, the simulation proceeds to the calculation of the nuclear burnup solution. If the k_{eff}^{analog} is out of bounds *and* iterations remain in the criticality search, ADER will build and solve the optimization problem for all material clusters, again. Following this the actions of discrete group-class streams only, discrete table-class streams are only applied on the very initial iteration of the criticality search, are again applied to materials to change their compositions. On every iteration of the criticality search after the initial iteration only the actions of discrete group-class streams are applied to the material compositions. The actions of discrete table-class streams are only applied on the initial iteration of a criticality search. If no criticality search is being done, if the system neutron multiplication factor has not been restricted, after the Monte Carlo transport sweep following the initial application of both discrete group and table-class streams the simulation calculation proceeds on to the burnup solution. To set the maximum number of criticality search iterations the input below is used, the default value of this parameter is 5. The criticality search features of ADER are only activated by the `kmax` and `kmin` keywords - i.e. a default value of 5 will not force a simulation through 5 criticality searches if no criticality search was requested.

```
set ader_trans_iter [value]
```

An important consideration of the interaction of streams with multiple forms, `cont`, `disc`, `prop`, is that if streams of multiple forms are used which significantly affect system reactivity ADER's criticality search will only be aware of the effects due to discrete streams. The effects on criticality from continuous and proportional streams are not assessed until the burnup calculation has been completed. For example if ADER has determined an amount of ^{233}U to inject into a material to keep it critical following a discrete injection of natural Li for chemistry control, and this uranium is from a continuous stream, the Monte Carlo transport sweep run before the burnup calculation will only see the negative reactivity effects of the discrete and instantaneous lithium injection. The positive reactivity effects of the uranium will be observed in the Monte Carlo transport sweep for the *next* burnup step.

B.6.1 Quick Reference

- The maximum system k_{eff}^{analog} is set by...

```
kmax [value]
```

- The minimum system k_{eff}^{analog} is set by...

```
kmin [value]
```

- The system defaults for `kmin` and `kmax` are zero and ∞ respectively. To use the system k_{eff}^{analog} as a constraint designate one and only one material with the keyword-value pair `rhow 1.0` in the material's definition and set appropriate `kmax` and `kmin` targets

- The maximum number of criticality search iterations is set by...

```
set ader_trans_iter [value]
```

- The criticality search feature of ADER is only aware of the reactivity impacts of discrete form streams, both group and table-class.

- When a criticality search is activated, the reactivity impacts *of all streams* are assessed as part of the constraints regarding the material designated by the keyword-value pair `rhow 1.0`.

B.7 Output

ADER output is located in the "[`input_file_name`]_dep.m" output file which SERPENT2 produces. Every material under ADER control with groups assigned via a conditions block or streams, will have output in this file.

B.7.1 Groups

The fraction of a material's atomic density accounted for by each group is printed out in a vector with the name of "MAT_m_GRP_g_FRAC" where m is the material name and g is the group name for a group which is a member of a material's options list - see section B.3 for a description of this list. The value at index j of such a vector corresponds to the value at the end of burnup step j. The values displayed are normalized to the material's atomic density at the beginning of the burnup step.

For example, consider if the group gFLiBe accounted for 60% of the material FuelSalt's atomic density during each of two burnup steps. The corresponding output line in the "_dep.m" file would appear as below...

```
MAT_FuelSalt_GRP_gFLiBe_FRAC = [
4.00000E-01, 4.00000E-01 ];
```

Summation groups and their component groups are all reported, in no particular order in the output.

B.7.2 Streams

The mass moved by a stream on each burnup step j is given in vector form where the vector is named "MAT_m_STREAM_s_STEP_AMT" where m is the material name and s is the name of the group or removal table used to define the group-class or table-class stream in question, respectively. Results for streams with both a valid sink and source are reported by each the sink and the source. Each vector index j corresponds to burnup step j. The units for stream output are $\frac{\texttt{atoms}}{barn \cdot cm}$ where the value represents the stream's impact on a per unit volume basis with the normalizing volume being the material volume for which the stream is reported. This means that if a given stream moves mass between materials of differing volumes, each material will report a different value for that stream for each burnup step because their volumes are different - all thanks to the conservation of mass. In the same context of atoms from section B.3, atoms in this sense refers to a fractional mix of isotopes corresponding to the recipe of the stream in question. For group-class streams these "atomic" units will have the proportions that the group recipe does - this is not necessarily the case for proportional group-class streams which have a bevy of cautions regarding their use presented in section B.5.1.3. For table-class streams these "atomic" units take the proportional make up of the table used to define the stream except in the case of proportional table-class streams in which case the "atoms" reported is just that, total atoms. The exact composition of such a stream is not given. Proportional type streams do report a true value, in the sense that the number of atoms is correct, in that the value accounts for nuclear depletion effects. This value is calculated inside the burnup calculation as what is sometimes referred to as a "ghost" nuclide - a false nuclide who's only impact on the solution is to track and measure some parameter of interest, in this case the true amount of mass moved by a proportional stream, group-class or table-class.

As an example, consider if stream `gUF4` had moved $0.001\frac{\texttt{atoms}}{barn \cdot cm}$ per unit volume into material `FuelSalt` on each of two burnup steps - so $0.0002\frac{\texttt{atoms}}{barn \cdot cm}$ of uranium and $0.0008\frac{\texttt{atoms}}{barn \cdot cm}$ of fluorine. The corresponding output line in the "_dep.m" file would appear as below...

```
MAT_FuelSalt_STREAM_gUF4_STEP_AMT = [
1.000000E-02, 1.00000E-02 ];
```

Summation streams and their component streams are all reported in the output in no particular order.

B.7.3 Quick Reference

- Group fractions of a material's atomic density, normalized to the material's atomic density at the beginning of the burnup step, are reported in a vector with the name of "`MAT_m_GRP_g_FRAC`" where `m` is the material name and `g` is the group name.
- Stream mass transfers are reported on a per unit volume basis of the associated material and are reported in the vector named "`MAT_m_STREAM_s_STEP_AMT`" where `m` is the source or sink material and `s` is the name of the group or removal table used to define the stream. The units are $\frac{atoms}{barn \cdot cm}$.

B.7.4 Developers Output

In the advanced compilation options for ADER, `ADER_DIAG` and `ADER_INT_TEST`, numerous additional outputs are produced. Any person who is making use of these outputs is expected to be familiar with the documentation of ADER as well as the source code. As such, only a brief summary of the additional files is given for each option. A more thorough understanding may be had by inspection of the functions which produce these outputs.

B.7.4.1 ADER_INT_TEST

Compiling SERPENT2 and ADER with the "`-DADER_INT_TEST`" flag will cause the following outputs to be produced. In the file name templates given `m` stands for a material name and `n` is a material cluster number.

- "ADER_Clp_Model_Material_m_Conformity.test" - contains an element by element comparison of the simplex model in ADER memory with the simplex model in CLP memory.
- "ADER_Cluster_Composition_Matrices.json" - contains all the information needed to construct the simplex problem in a json format.
- "Cluster_n_Material_Composition_Matrix.csv" - contains every entry of the simplex matrix for material cluster `n`.

B.7.4.2 ADER_DIAG

Compiling SERPENT2 and ADER with the "`-DADER_DIAG`" flag will cause the following outputs to be produced. In the file name templates given `m` stands for a material name, `s` stands for the numeric index of a given burnup step, `ss` stands for the numeric index of the sub-step for burnup step `s`, `n` is a material cluster number, and `k` is the index of the criticality search iteration.

- "Cluster_Parent_m_Burn_Matrix_Step_s_Sub_Step_ss.csv" - contains a copy of the burnup matrix for the material cluster who's parent is material `m` for the given burnup step and sub-step.
- "Cluster_Parent_m_Pred_Step_s_Sub_Step_ss_Matrix_Comparison.test" - contains a comparison of every compatible entry in the burnup matrix as generated by SERPENT2 and ADER for the prediction step of a predictor-corrector burnup simulation.
- "Cluster_Parent_m_Corr_Step_s_Corr_Sub_Step_ss_Matrix_Comparison.test" - contains a comparison of every compatible entry in the burnup matrix as generated by SERPENT2 and ADER for the corrector step of a predictor-corrector burnup simulation.
- "ADER_Memory_Lists.test" - contains `WDB` address information for a variety of data.
- "m_XS_End_of_Step_s.txt" - contains a comparison of the isotopic cross sections (absorption and fission) between SERPENT2 and ADER after all criticality search iterations for a given burnup step.
- "m_XS_step_s_iter_k.txt" - contains a comparison of the isotopic cross sections (absorption and fission) between SERPENT2 and ADER for the given burnup step `s` and inner criticality iteration `k`.

B.8 Testing

ADER comes complete with a unit test suite and a suite of system tests.

B.8.1 Unit Tests

To run ADER's unit tests, compile the code with the "`-DADER_TEST`" flag and run, on a single thread, the input file "`test_input.txt`" found in the "`inputs/Test_Input`" directory with the run option "`-test`". The results of the unit tests will be found in the executing directory in a file titled "`TestResults.test`".

B.8.2 System Testing

To run any one of ADER's system tests compile the code anyway except with the "`-DADER_TEST`" flag. Then, with any number of threads, run any of the system test files found in any of the system test directories found in the "`System_Testing`" directory. Compare the results with the desired results described in the README files.

B.9 Parallel Computation

ADER has been developed for threaded operation with the OpenMP interface just as SERPENT2 has. To run ADER with more than one thread do the same thing SERPENT2 asks, add the input seen below to the command line executing the SERPENT2 run. This will enable threaded computation for SERPENT2 as a whole as well, and this input does not need to be duplicated for both SERPENT2 and ADER to run in a threaded manner - only one instance of this command line option should be used.

```
-omp [num_threads]
```

ADER is not compatible with distributed memory computing - i.e. while SERPENT2 can do MPI runs, SERPENT2 with ADER can not. The code as a whole may be compiled in a way to run with MPI but only non-ADER simulations can be run with MPI.

B.10 General Notes

A few closing notes, restrictions, hints, and tips that didn't fit elsewhere in the manual...

- Any SERPENT2 material with either a conditions block or one or more streams, must have the keyword `ader` appear in the material's definition before the listing of the material's constituent isotopes.
- If the user would like ADER to issue a warning every time the burnup solution produces a negative density for an isotope, which is then corrected to zero, place the following line in the simulation input. The value is arbitrary.

  ```
  set ader_neg_adens [value]
  ```

- ADER is not compatible with divided SERPENT2 materials. These may be used in the same simulation but divided materials, or their children, may not also be materials under ADER control.
- ADER is not compatible with SERPENT2's "`mflow`" structure. Materials affected by `mflow` structures may be used in an ADER simulation but they may not in any way (other than neutronically) be connected with ADER or ADER materials.

- SERPENT2 employs a function, `MaterialBurnup`, which calculates the power produced by a material during a burnup step by the difference in abundances of isotopes in that material between the beginning of the burnup step and the end of the burnup step. This methodology assumes there are no mass flows in the system and has not been configured to incorporate ADER mass flows. AS SUCH - MATERIAL POWER ESTIMATES PROVIDED BY SERPENT2 WHEN RUNNING AN ADER SIMULATION WITH AT LEAST ONE CONTINUOUS OR PROPORTIONAL STREAM ARE LIKELY INCORRECT. The user can fix these power estimates themselves by incorporating the mass moved by streams and the power produced from any fissions on these masses.
- ADER's oxidation table structure does not recognize elements higher than Rg on the periodic table.
- On EVERY Monte-Carlo sweep SERPENT prints out cycle-wise data into the `[name]_res.m` file. Every ADER iteration will produce a new entry for the same burnup step in this file with older entries becoming invalid but not erased. Be aware.

B.11 Theory

The details of ADER's computations and the theory behind its operation are laid out in this section.

B.11.1 Groups

Take $F_{e|u}$ to be the input values given for each element, e, in a group, u. Take $F_{i|e,u}$ to be the input value given for each isotope, i, as part of element e in group u. The normalized equivalents, $f_{e,u}$ and $f_{i|e,k}$, are defined by equations B.16 and B.17 respectively.

$$f_{e,u} = \frac{F_{e,u}}{\sum_e^E F_{e,u}} \tag{B.16}$$

$$f_{i|e,u} = \frac{F_{i|e,u}}{\sum_{i|e}^{I|e} F_{i|e,u}} \tag{B.17}$$

Concerning the elements in groups which are not given an explicit isotopic composition, referred to as "unfixed" elements - calculations in SERPENT2 are ultimately done on an isotopic basis and so all elements must be composed of isotopes. Groups which are a part of a material's options list, see section B.3 for an explanation of this term, derive the isotopic compositions for their unfixed elements using the isotopic composition of the same element in the host material. These $f_{i:e,u}$ values are derived as seen in equation B.18 where $N_{i:e}$ is the number density of isotope i of element e in the material of interest.

$$f_{i|e,u} = \frac{N_{i|e}}{\sum_{i|e}^{I|e} N_{i|e}} \tag{B.18}$$

Take $f_{i,u}$ to be the normalized fraction of isotope i in group k. These values are derived as seen in equation B.19.

$$f_{i,u} = f_{e,u} f_{i|e,u} \tag{B.19}$$

Take g_u to be the normalized fraction of a material's atomic density which is taken to ascribe to the recipe for group u - that is, the isotopes composing this subset of the material density all have a normalized abundance of $f_{i,u}$ within the subset. For instance, consider the atomic density and makeup of the material `FuelSalt` from section B.2 and consider the group `gFLiBe` from section B.3. The g_u value for `gFLiBe` in the material `FuelSalt` could be said to be anywhere from zero to 0.76.

Take E_j to be the number density of element j in a material. Take I_k to be the atomic density of isotope i in a material. Take μ_j to be the portion of element j's number density not attributed to group structures. Take μ_k to be the portion of isotope k's atomic density which is not attributed to group structures. In the case of a "controlled" element or isotope, a concept covered in section B.4.3, the μ value would be zero. The relationships then seen in equations B.20 and B.21 may be established.

$$E_j = \mu_j + \sum_{u}^{U} g_u f_{e,u} \tag{B.20}$$

$$I_k = \mu_k + \sum_{u}^{U} g_u f_{i,u} \tag{B.21}$$

In section B.4.2 the concept of a relative abundance constraint between two groups was developed. Equation B.4 can be expressed linearly as two inequalities as shown in equations B.22 and B.23 where r_m is the input `min_value` and r_M is the input `max_value`.

$$-\infty \leq -g_1 + r_m g_2 \leq 0 \tag{B.22}$$

$$0 \leq -g_1 + r_M g_2 \leq \infty \tag{B.23}$$

Summation groups, those groups defined according to method D in section B.3, have their g_k values - denoted $g_{k:Y}$ - determined according to equation B.24 where $g_{k|Y}$ is the g_k value for a group used in the definition of summation group Y and $w_{k|Y}$ is the weight value entered for group k used in the definition of summation group Y. In section B.3 this value is shown as a mandatory input of "`1`" though in reality it may take any value greater than 0 - but any value other than 1 may lead to a non-physical solution in the optimization problem.

$$g_{k:Y} = \sum_{k|Y}^{K|Y} g_{k|Y} w_{k|Y} \tag{B.24}$$

B.11.2 Group-class Streams

For group-class streams take s_v to be the atomic density of this mass load where an "atomic" unit is composed of isotopes in proportion to the $f_{k,u}$ values of the group, which are denoted $f_{k,v}$ for groups which are used to form streams.

Take $E_{j,\Delta}$ to be the number density of element j in a given stream's mass load. Take $I_{k,\Delta}$ to be the atomic density of isotope i in a given stream's mass load. The relationships then seen in equations B.25 and B.26 may be established.

$$E_{j,\Delta} = s_v f_{e,v} \tag{B.25}$$

$$I_{k,\Delta} = s_v f_{i,v} \tag{B.26}$$

Concerning the elements in streams which are not given an explicit isotopic composition, referred to as "unfixed" elements - calculations in SERPENT2 are ultimately done on an isotopic basis and so all elements must be composed of isotopes. Streams derive the isotopic compositions for their unfixed elements using the isotopic composition of the same element in the source material. These $f_{i|e,v}$ values are derived as seen in equation B.27 where $N_{i|e}$ is the number density of isotope i of element e in the source material of interest. Streams with unfixed elements and no SERPENT2 material as a source are not permitted.

$$f_{i|e,v} = \frac{N_{i|e}}{\sum_{i|e}^{I|e} N_{i|e}} \tag{B.27}$$

Summation streams, group-class streams constructed with summation groups have their s_v values - denoted $s_{v:Y}$ - determined according to equation B.28 where $s_{v|Y}$ is the s_v value for a stream used in the definition of summation stream Y and $q_{v|Y}$ is the weight value entered for group v used in the definition of summation stream Y. In section B.3 this value is shown as a mandatory input of "`1`" though in reality it may take any value greater than 0 - but any value other than 1 may lead to a non-physical solution in the optimization problem.

$$s_{v:Y} = \sum_{v|Y}^{V|Y} s_{v|Y} q_{v|Y} \tag{B.28}$$

B.11.3 Table-class Streams

The change in an isotope's, i, atomic density, $z_{i,v}$, induced by a continuous or discrete table-class stream, v is given by equation B.29 where $q_{i,v}$ is the `table_value` given for isotope i

by the removal table used to create stream v, N_i is the number density for that isotope, and r_v is the `frac_value` given for stream v.

$$z_{i,v} = q_{i,v} r_v N_i \tag{B.29}$$

The change in an isotope's, i, atomic density, $z_{i,v}$, induced by a proportional table-class stream, v is given by equation B.30 where $q_{i,v}$ is the `table_value` given for isotope i by the removal table used to create stream v, N_i is the number density for that isotope, and r_v is the `frac_value` given for stream v. If the proportional stream in question, v, has only a single valid SERPENT2 material as a sink, the sign of $q_{i,v}$ is positive. Otherwise, the sign of $q_{i,v}$ is negative.

$$z_{i,v} = N_i e^{q_{i,v} r_v \Delta t} \tag{B.30}$$

The total change in an isotope's, i, atomic density due to all tale-class streams is denoted r_i is given by equation B.31. It should be noted, that when this summation involves values for a proportional stream the r_i value becomes an approximation as the proportional stream will move more or less mass based on nuclear depletion and the action of other streams.

$$r_i = \sum_{v}^{V} z_{i,v} \tag{B.31}$$

There are of course $z_{e,v}$ and r_e equivalents for elements as well.

B.11.4 Criticality Control

A weighted sum over isotopes in a material forms the reactivity constraint that may be applied. This constraint is derived from the expression for the multiplication factor as found in Equation B.32:

$$k_{eff} = P_{NL} \frac{\sum_{m}^{M} \phi_m \omega_m \nu \Sigma_f^m}{\sum_{m}^{M} \phi_m \omega_m \Sigma_a^m} \tag{B.32}$$

where ϕ_m, ω_m, $\nu\Sigma_f^m$ and Σ_a^m are, respectively, the scalar neutron flux, the volume fraction, the spectrum averaged neutron production cross section, and the macroscopic absorption cross section for each material m. P_{NL} is the neutron non-leakage probability. In ADER, the ability to control k_{eff} is limited to the case of a single neutron multiplying material; take $\nu\Sigma_f = 0$ for every material but the multiplying material M and as such Equation B.32 can be rewritten as follows:

$$k_{eff} = P_{NL} \frac{\phi_M \omega_M \Sigma_a^M}{\sum_{m}^{M} \phi_m \omega_m \Sigma_a^m} \frac{\nu \Sigma_f^M}{\Sigma_a^M} \tag{B.33}$$

The probability of a neutron being absorbed in the multiplying material is defined as follows:

$$P_A = P_{NL} \frac{\phi_M \omega_M \Sigma_a^M}{\sum_m^{M-1} \phi_m \omega_m \Sigma_a^m} \tag{B.34}$$

Then k_{eff} can be calculated as:

$$k_{eff} = P_A \frac{\nu\Sigma_f^M}{\Sigma_a^M} = P_A \frac{\sum_i^I \nu\Sigma_f^i}{\sum_i^I \Sigma_a^i} \tag{B.35}$$

where $\nu\Sigma_f^i$, and Σ_a^i are, respectively, the spectrum averaged neutron production cross section, and the absorption cross section for every isotope i in the multiplying material M. This relation is expected to hold for simulations in which there is a dominant reactive material and for which $\nu\Sigma_f \approx 0$ for all other materials.

In this case, given lower and upper bounds for the multiplication factor of the system, k_{eff}^{min} and k_{eff}^{max} respectively, Equation B.35 can be made linear as in Equations B.36 and B.37.

$$0 \geq \frac{k_{eff}^{min}}{P_A} \sum_k^K \sigma_a^k I_k - \sum_k^K \nu^k \sigma_f^k I_k \tag{B.36}$$

$$0 \leq \frac{k_{eff}^{max}}{P_A} \sum_k^K \sigma_a^k I_k - \sum_k^K \nu^k \sigma_f^k I_k \tag{B.37}$$

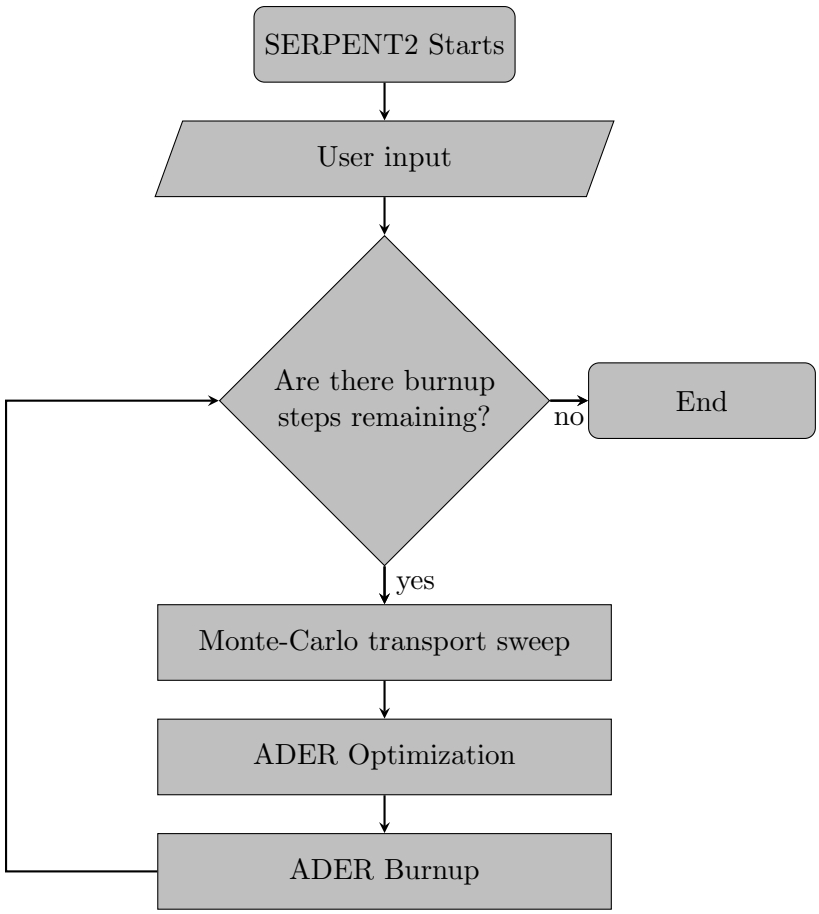
A key assumption of this linearization process is that $\frac{\partial P_A(m...M)}{\partial M} = 0$ when in truth P_A is a function of the composition of material M. The impacts of this approximation are expected to be quite small but it will affect all simulations, more so those with strong leakage effects.

B.11.5 Constructing the Optimization Problem

A discussion of the construction of the optimization problem would not be complete without a discussion of ADER's interactions with SERPENT2 and a view of the overall program flow. Figure **??** provides a simplified view of ADER's interactions with SERPENT and the overall simulation progression. In figure **??** the "ADER Optimization" box from figure **??** is expanded as a process into its own flowchart.

In a SERPENT2 burnup simulation, following the initial Monte Carlo transport sweep done at the beginning of every burnup step, ADER enters its criticality search iterations - even if a criticality search has not been asked for. The default values for `kmin` and `kmax` will cause the criticality check to pass regardless.

Figure B.1: A simplified schematic of interactions between SERPENT2 and ADER.



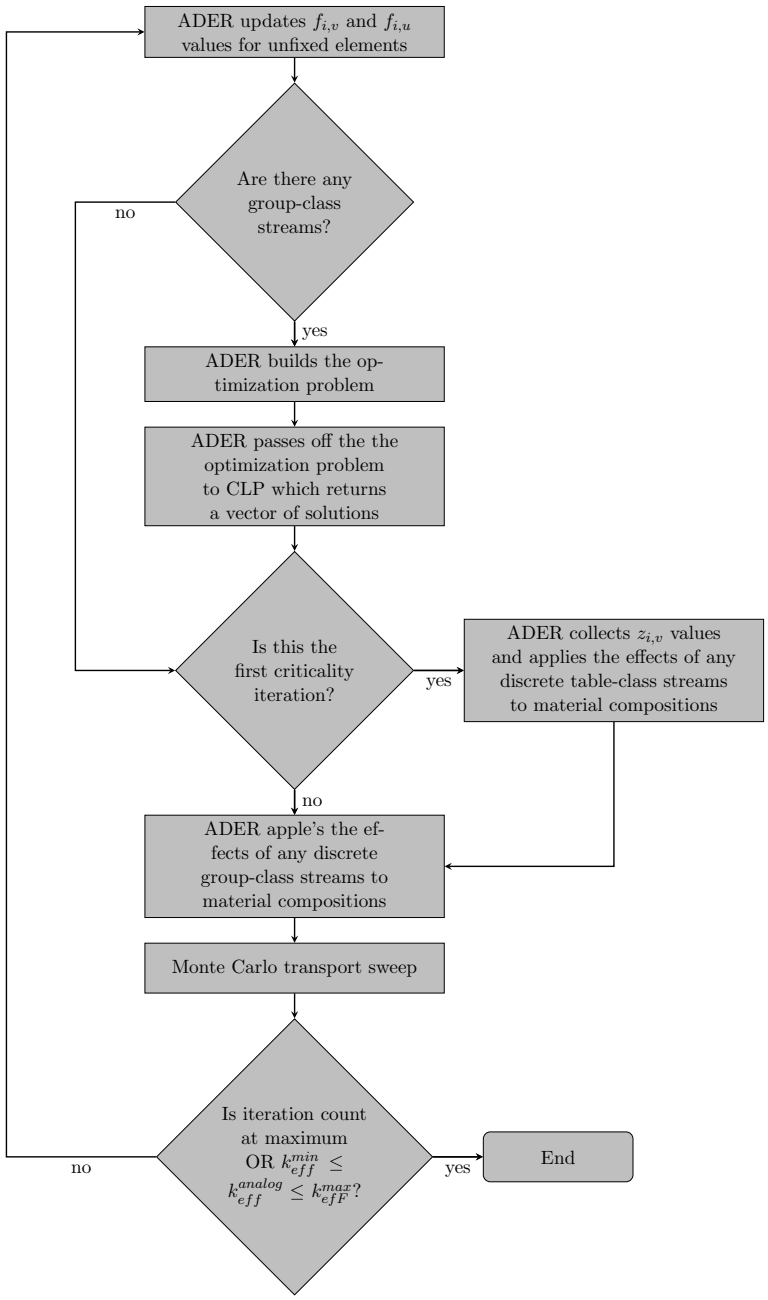
The first consequential action ADER takes in each iteration is to determine the $f_{i|e,u}$ and $f_{i|e,v}$ values for the isotopes of unfixed elements in groups and group-class streams. Following this assessment all of the $z_{i,v}$ values are calculated for table-class streams.

This point marks the first major divergence of any ADER simulation. *If and only if* the only streams entered by the user are table-class streams the simulation proceeds to the application of the effects of discrete form streams. There is no optimization process in this case because the user has only given direct orders to ADER in the form of table-class streams, there are no choices for ADER to make in the form of group-class streams.

In the case that at least one group-class stream, attached to a material, exists in the simulation, the simulation would then proceed to build and solve the optimization problem for each material cluster.

The CLP library expects a linear programming matrix from ADER. Figure B.11.5 depicts the scheme for constructing the linear programming matrix. Column bounds are presented above the appropriate column whereas row bounds are presented to the left of the appropriate row. Below the column bounds are the variables which the columns represent and to the right

Figure B.2: A detailed look of the "ADER Optimization" box from figure **??**. This diagram picks up at the entry to the "ADER Optimization" box and exits out to the "ADER Burnup" box also in figure **??**.



of the row bounds are the equation number, if any, of the equation that row is modeled after. For the sake of brevity the matrix in Figure B.11.5 is for one material only though many materials may be involved in such a matrix should they be linked together by shared mass transfers. In which case the only variables shared between materials in the same material cluster are the group-class streams and the stream equations they are a part of are the only coupling equations; aside from transfers by table-class streams but those are only represented in the linear programming matrix, they are handled by other routines all together. If a second material were to be included in this matrix then, perhaps, the stream entries in the fourth, fifth, and sixth columns would have non-zero coefficients for some E_j^d and I_k^d rows of the second material. The coefficients used for the construction of these matrices are normalized to the atomic density from the beginning of the optimization process for the host material.

Working down the matrix row by row the first row encountered represents equation B.22 with arbitrary groups g_1 and g_2 whereas the next row down represents equation B.23. The third row, what will be referred to as an elemental future row represents the atom balance for element j where fe_j^u is the fractional proportion of element j in group u. The novel column involved here is an elemental future column whose inclusion in the same row closes the equation where an "elemental future value", E_j^f, is the fraction of a material's atomic density (relative to the density at the beginning of the step) that is taken up by element e_j. The bounds for this row are those for a free element, a concept covered in section B.4.3, those elements which are permitted to have portions of the element not tied up in declared group structures. In the case of a controlled element, those who's complete abundance must be accounted for by group structures, the lower bound is changed to zero. The fourth row, an elemental delta row, represents the change in the abundance of element j, E_j^d, as caused by all group-class streams where fe_j^v is the fractional proportion of element j in stream v. Of course the elemental delta column is involved to close the balance. The fifth row, or balance row, is what ties together E_j^f and E_j^d. The bounds, α and β, are equal and represent $E_j^c + r_{e_j}$ constituting an element balance "in time where E_j^c represents the present fractional abundance of element j. The fifth row requires, straightforwardly, that the future amount of an element be equal to the current amount plus any delta, or change, in the element's abundance. The sixth row is an isotopic balance row requiring that the abundance of an element be equal to the abundance of its constituent isotopes. The following three rows, the seventh, eighth, and ninth, are the isotopic versions of the elemental future, delta, and balance rows where f values are for the isotopic fractional proportions. γ and δ are equal and represent $I_k^c + r_i$ where I_k^c represents the present fractional abundance of isotope k. In the tenth and eleventh rows Equations B.37 and B.36 find representation with η and θ respectively representing terms of the expanded sum found in the referenced equations; $\frac{k_{eff}^{min}}{P_A}\sigma_a^k - \nu^k\sigma_f^k$ and $\frac{k_{eff}^{max}}{P_A}\sigma_a^k - \nu^k\sigma_f^k$. The twelfth row represents equation B.6, accounting for the contributions of the future quantity of an element to a material's averaged oxidation state (or however the weighted sum is interpreted). The thirteenth row, or Pres row, exists when the user instructs ADER to balance inflows with outflows as discussed in section B.4.5. The pres row requires that the net stream transfers in a material come to zero. The effects

of table-class streams are captured in υ and ω as seen in equation B.38. The fourteenth row represents a closure for a group defined with method D from section B.3; a summation group. In this case the summation group is g_3 which was defined as follows...

```
grp  g3  sum
g1   w1
g2   w2
```

The fifteenth row represents a closure relationship for a summation stream, a group-class stream built using a summation group. In this case the summation stream is s_3. The final row is the optimization, or Opt row. This row indicates to the simplex routine which variables to minimize or maximize. In figure B.11.5 the opt row is indicating that g_1 is the optimization target.

$$\upsilon = \omega = \sum_{i}^{I} r_i \tag{B.38}$$

		$[b_m, b_M]$ g_1	$[0,\infty)$ g_2	$[0,\infty)$ g_3	$[0,\infty)$ s_1	$[0,\infty)$ s_2	$[0,\infty)$ s_3	$[0,\infty)$ E_j^f	$(-\infty,\infty)$ E_j^d	$[0,\infty)$ I_k^f	$(-\infty 0,\infty)$ I_k^d
$(-\infty,0]$	Eq.$B.22$	-1	r_m								
$[0,\infty)$	Eq.$B.23$	-1	r_M								
$(-\infty,0]$	E_j^f	$f_{e_j^1}$	$f_{e_j^2}$					-1			
$[0,0]$	E_j^d				$f_{e_j^1}$	$f_{e_j^2}$			-1		
$[\alpha,\beta]$	E_j^b							1	-1		
$[0,0]$	E_j^i							-1		1	
$(-\infty,0]$	I_k^f	$f_{i_k^1}$	$f_{i_k^2}$							-1	
$[0,0]$	I_k^d				$f_{i_k^1}$	$f_{i_k^2}$					-1
$[\gamma,\delta]$	I_k^b									1	-1
$[0,\infty)$	Eq.$B.37$									η	
$(-\infty,0]$	Eq.$B.36$									θ	
$[O_m,O_M]$	Eq.$B.6$							$v_e w_e$			
$[\upsilon,\omega]$	Pres				1	1					
$[0,0]$	Eq.$B.24$	$-w_{1\|3}$	$-w_{2\|3}$	1							
$[0,0]$	Eq.$B.28$				$-q_{1\|3}$	$-q_{2\|3}$	1				
	Opt	1									

B.11.6 Solving the Optimization Problem

Once the matrix seen in figure B.11.5 has been constructed by ADER it is converted into a dense column-major format and passed off to the CLP library which solves this matrix as a simplex problem. CLP then returns a vector containing the atomic density of each group in a material, normalized to the material's density at the beginning of the optimization process. This vector also contains all the group-class mass load values needed to bring the material cluster to the optimal state. If the simulation is in the first criticality search iteration for a burnup step, the actions of all discrete form streams,both group-class and table-class,

are applied to materials. If the simulation is past the first criticality search iteration for the current burnup step only the actions of discrete group-class streams are applied to materials. Following these actions a Monte Carlo transport sweep is run with all the same cycle and batch parameters as specified by the user in the SERPENT2 input files. At this point, if the system k_{eff}^{analog} is within the bounds as set by the user (or the default bounds) or if the criticality search has already used up all of its iterations, the simulation progresses on to the burnup calculation. If not, another criticality search iteration is launched starting with the determination of the isotopic composition for all unfixed elements in groups and group-class streams.

B.11.7 Nuclear Burnup Calculations

ADER's burnup routine rides along the iteration scheme employed by the SERPENT2 burnup routine. That is to say that ADER is compatible with any burnup correction scheme that the user employs through SEREPNT2. In truth, the only affect that a burnup iteration scheme has on ADER is to change the cross sections used in any criticality control rows in the optimization matrices. ADER extends the burnup capabilities of SERPENT2 to include the effects of both group and table-class streams. The coefficients in the burnup matrix are those from the Bateman equation as seen in equation B.39 for the one energy group and zero dimensionality case, where N is the number density of nuclide n, t is time, $b_{m\to n}$ is the branching ratio for the decay of nuclide m into n, λ is the decay constant for its sub-scripted nuclide, q goes over all neutron induced absorption reactions for a given isotope, $a^q_{m\to n}$ is the branching ratio for isotope m into n due to reaction q, σ^y_x is the effective microscopic cross section of reaction x for isotope y, ϕ is the scalar neutron flux, d denotes all transmutation reactions for a given isotope, $R_n(t)$ is a fractional removal (or addition) rate for isotope n at time t, and $F_n(t)$ is a feed (or removal) amount for isotope n at time t. These last two terms in equation B.39 account for proportional and continuous form streams respectively.

A highly truncated burnup scheme can be seen in figure B.40 in which there are two isotopes, ^{233}U and ^{135}Xe, and two streams; S_c representing a continuous stream with a constant injection rate and S_p representing a proportional stream with a transfer rate dependent upon the concentration of the substances to be transferred. There are, of course, two matrices as well. The burnup matrix to the left holding the coefficients of the Bateman equation and the second, to the right, holding the initial concentrations of isotopes and the values for the streams. The first column of the first row gives the creation and destruction of ^{233}U which is dependant on the concentration of ^{233}U with Γ representing nuclear destruction as seen in equation B.41. The third column of the first row holds the fraction of stream S_c that ^{233}U comprises. These entries together describe the evolution of ^{233}U in the given system. In the second row Ξ, as seen in equation B.42, represents the production of ^{135}Xe from ^{233}U. In the second column of the second row are the processes dependant on the concentration of ^{135}Xe. Υ represents the proportional rate constant as determined by the multiplication of $q_{i,v}$ and r_v whereas Θ is given by equation B.43. The third row is blank as the abundance of a continuous type stream, c_n, does not change over a burn step. The fourth row is an

addition specific to ADER and not found in the Bateman equations; rather, this line, and the lines it represents, exists to keep track of the amount of an isotope that a proportional stream moves simply to provide this information to the user. The system of matrices seen in figure B.40 is solved by SERPENT2 using the CRAM methodology providing updated isotopic abundances and proportional stream transfer amounts.

$$\begin{aligned}\frac{\mathrm{d}N_n(t)}{\mathrm{d}t} =& \sum_{m}^{M} b_{m\to n}\lambda_j N_j(t) + \\ & \sum_{m}^{M}\sum_{q}^{Q} a^q_{m\to n}\sigma^k_q\phi(t)N_k(t) - \\ & N_n(t)\lambda_i - \sum_{d}^{D}\sigma^n_d\phi(t)N_n(t) - \\ & R_n(t)N_n(t) + F_n(t)\end{aligned} \tag{B.39}$$

$$\begin{array}{c} \\ {}^{233}\mathrm{U} \\ {}^{135}\mathrm{Xe} \\ S_c \\ S_p \end{array} \begin{array}{c} \begin{array}{ccccc} {}^{233}\mathrm{U} & & {}^{135}\mathrm{Xe} & S_c & S_p \end{array} \\ \begin{bmatrix} -\lambda_{^{233}\mathrm{U}} + \Gamma & & f^{S_c}_{^{233}\mathrm{U}} & \\ \Xi & -\lambda_{^{135}\mathrm{Xe}} + \Upsilon + \Theta & & \\ & & & \\ & \Upsilon & & \end{bmatrix} \end{array} \begin{array}{c} \mathbb{N} \\ \begin{bmatrix} N_{^{233}\mathrm{U}} \\ N_{^{135}\mathrm{Xe}} \\ c_n \\ 0 \end{bmatrix} \end{array} \tag{B.40}$$

$$\Gamma = -\sum_{d}^{D}\sigma_d^{^{233}\mathrm{U}}\phi \tag{B.41}$$

$$\Xi = b_{^{233}\mathrm{U}\to^{135}\mathrm{Xe}}\lambda_{^{233}\mathrm{U}} + \sum_{q}^{Q} a_{^{233}\mathrm{U}\to^{135}\mathrm{Xe}}\sigma_q^{^{233}\mathrm{U}}\phi \tag{B.42}$$

$$\Theta = -\sum_{d}^{D}\sigma_d^{^{135}\mathrm{Xe}}\phi \tag{B.43}$$

Following the solution of the burnup problem the material compositions are updated accordingly and the simulation moves on to the next burnup step.

B.12 Installation

Installing ADER is a five step process: Downloading ADER covered in section B.12.1, installing the CLP libraries covered in section B.12.2, editing the necessary lines in the SERPENT2 base code covered in section B.12.3, compiling the code covered in section B.12.4,

and testing the code, already covered in section B.8. These instructions assume the user is operating inside of a linux-like environment.

B.12.1 Downloading ADER

ADER is available as a public repository hosted at...

some_web_address

Download ADER using your preferred client. The directories contained in the ADER parent directory, call this folder `ADER_Dir`, are...

- `docs` - where this user manual and the API can be found in the subdirectories `docs/UM` and `docs/API`, respectively.
- `inputs` - where in the subdirectory, `inputs/Test_Input`, the file `test_input.txt` can be found.
- `src` - where all of ADER's source files can be found.
- `System_Testing` - where all of ADER's system tests can be found in their respective subdirectories titled after the test which they contain. Inside of each subdirectory is the test input file and a README file explaining the operation of the test.

B.12.2 Downloading and Installing CLP

CLP is available as a public repository hosted at...

https://github.com/coin-or/Clp

Download CLP using your preferred client. Inside this directory, call it `Clp_Dir`, there is one important subdirectory, `Clp_Dir/Clp`. Inside of this subdirectory executing the following commands in a linux-like environment should install the Clp libraries on your system...

```
./configure
./install-sh
make all
```

Copy the file `Clp_C_Interface.h`, found in `Clp_Dir/Clp/src`, into your SERPENT2 build directory. After this process is complete do not forget to add the path to this subdirectory to all applicable system paths, most likely just your system `PATH`.

B.12.3 Editing SERPENT2

These instructions assume that the base version of SERPENT2 being modified is version 2.1.31. ADER DOES NOT WORK WITH VERSIONS OF SERPENT2 EARLIER THAN 2.1.30. Compatibility with future versions of SERPENT2 and installation directions are

not guaranteed. To configure SERPENT2 to interact with ADER three steps are necessary. First, copy all of the files found in `ADER_Dir/src`, to the SERPENT2 `src` directory (or wherever you build SERPENT2). This might look something like the following...

```
cd ~/SERPENT2_Dir
cp ~/ADER_Dir/src/* ./src/
```

In the directory `ADER_Dir/docs/UM` there is a file named "Source_mod.txt". "Ln" is short for "line number". The contents of this file have the following format...

```
[function_name].c: Line number or description of location in file
{
#ADER MOD BEGIN#
contents to add to file at the designated location
#ADER MOD END#
}
```

For each listing, add the code contents found between the `ADER MOD BEGIN` and `ADER MOD END` tags to the actual SERPENT2 source file given at the location described: this is either inserting new code lines following an existing SERPENT2 source line given as either an insertion listing or as a single line number or as a range of line numbers in which case this range is to be fully replaced by the ADER code sample. A suggestion is to execute these file modifications from the modifications closest to the end of the file to the modification closest to the beginning of the file - in this way the line numbers you look for are unchanged by the addition of the ADER code. It is considered good practice to include the tags themselves ,`ADER MOD BEGIN` and `ADER MOD END`, but this is not strictly necessary. Consider the file, "main.c", seen below...

```
#include header.h

void main(args)
{
long j;
long i = 0;

i = 1;

return(i);
}
```

Now consider that one of the entries in the file "Source_mod.txt" looks like the below...

```
Main: Ln 6-8
{
#ADER MOD BEGIN#
for(j=0; j<10; j++)
```

```
{
    i++;
}
#ADER MOD END#
}
```

The correctly modified main.c file would look like the below where lines 6 through 8 of the base file were replaced with the content between and including the ADER mod tags: "`ADER MOD BEGIN`" and "`ADER MOD END`". Please use your best discretion when editing these files. If a line number direction would cut off a `for` loop and cause compilation or logic errors, obviously that line number is incorrect and should be slightly different.

```
#include header.h

void main(args)
{
long i = 0;
#ADER MOD BEGIN#
for(j=0; j<10; j++)
{
    i++;
}
#ADER MOD END#
return(i);
}
```

B.12.3.1 Modifying the Makefile

Add the following lines to the SERPENT2 Makefile before the `OBJS` list but replace `[absolute/path/to/the/CLP/library]` with the actual path to the Clp library on your system.

```
# ADER MOD BEGIN #
################################################################################

# Below should be "-L[absolute/path/to/the/CLP/library] -l
#     and -L[absolute/path/to/the/CLP/library] -lclp

LDFLAGS += -L/Clp_Dir/Clp/lib -lClpSolver -L/Clp_Dir/Clp/l

# Enable ADER_TEST to run the test_input.txt file to execu
#     unit and integration test suite

#CFLAGS += -DADER_TEST
```

```
# Enable ADER_INT_TEST to ouput the files necessary for in
#      testing. Many of these files are human readable and
#      for manual debugging

#CFLAGS += -DADER_INT_TEST

# Enable ADER_DIAG to output a copious amount of debugging

#CFLAGS += -DADER_DIAG

##############################################################################
# ADER MOD END #
```

In the directory `ADER_Dir/docs/UM` there is a file named "Object_list.txt" - add the contents of this file to the SERPENT2 Makefile `OBJS` list. You may alphabetize the entires if you care to but it is not necessary.

In the directory `ADER_Dir/docs/UM` there is a file named "Build_list.txt" - add the contents of this file to the SERPENT2 Makefile build list - the long list of makefile build instructions at the bottom of the file. You may alphabetize the entries if you care to but it is not necessary.

B.12.4 Compiling ADER

To compile ADER with SERPENT2, after having followed the steps in all previous subsections of this section, give the command below inside of your `SERPENT2_Dir/src` directory...

```
make all
```

To compile ADER for unit testing, uncomment the line with the phrase "`CFLAGS += DADER_TEST`". When compiled this way the entire code will work for no other purpose than ADER unit testing conducted with the file "`test_input.txt`" found inside the "`ADER_Dir/inputs/Test_Input`" directory.

The compilation flags "`DADER_DIAG`" and "`DADER_INT_TEST`" are covered in section B.7.4.

Appendix C

Input for Chapter 3

The following, as given in the verbatim environment of LaTeX which utilizes the below font for distinction from "normal" text, is a direct reproduction of the code-input used for the simulations in chpater 3.

```
% ------ Created November, 26th, 2018, by Daniel D. Wooten
% ------ Comments
% ------ <<
% ------ >
% ------ >>

set title "TCO"

% --- Surfaces

% ------ Reactor Core
surf    1 cube   0 0   0   100

% --- Cells
cell    11      0       fuel           -1
cell    21      0       outside         1

% --- Materials

mat fuel -3.3052 vol 1E6 ader burn 0 cnd Cfuel rhow 1.0
3006.06c    0.00001421
3007.06c    0.28425216
4009.06c    0.06338586
9019.06c    0.60408583
90232.06c   0.04288894
92233.09c   0.00476985
```

```
% ---- Materials for detectors

mat dLi6    1.0 3006.06c 1.0
mat dLi7    1.0 3007.06c 1.0
mat dBe9    1.0 4009.06c 1.0
mat dC12    1.0 6012.06c 1.0
mat dF19    1.0 9019.06c 1.0
mat dTh232  1.0 90232.06c 1.0
mat dU233   1.0 92233.06c 1.0

% --- End materials for detectors

%--- Detectors
det 1   dm  fuel
det 2   dm  fuel    dr  -2  dLi6    dt  3   1
det 3   dm  fuel    dr  -2  dLi7    dt  3   1
det 4   dm  fuel    dr  -2  dBe9    dt  3   1
det 5   dm  fuel    dr  -2  dF19    dt  3   1
det 6   dm  fuel    dr  -2  dTh232  dt  3   1
det 7   dm  fuel    dr  -2  dU233   dt  3   1
det 8   dm  fuel    dr  18  dU233   dt  3   1
det 9   dm  fuel    de  lgrid   dt  -3
det 10   dm  fuel    dr  -2  dLi6    dt  3   1 de egrid
det 11  dm  fuel    dr  -2  dLi7    dt  3   1 de egrid
det 12  dm  fuel    dr  -2  dF19    dt  3   1 de egrid
det 13  dm  fuel    dr  -2  dTh232  dt  3   1 de egrid
det 14  dm  fuel    dr  -2  dU233   dt  3   1 de egrid
det 15  dm  fuel    dr  18  dU233   dt  3   1 de egrid

ene lgrid 3 200 1E-10   3
ene egrid 4 cas70

% --- End Detectors

% ------ Groups

grp gLi
Li  1

grp gLiS
Li  1

grp gAllLi  sum
```

```
gLi 1
gLiS 1

grp gLi6
Li  1 isos 1
Li-6    1

grp gLi7
Li  1 isos 1
Li-7    1

grp gBe
Be  1

grp gC
C   1

grp gCo
C   1

grp gF-li
F   1

grp gF-be
F   1

grp gF-th
F   1

grp gF-u
F   1

grp gTh232
Th  1   isos    1
Th-232  1

grp gU
U   1

grp gUF4i
U   1 isos 1
U-233 1
F   4   isos    1
F-19    1
```

```
grp gThF4i
Th   1 isos 1
Th-232 1
F   4   isos    1
F-19    1

grp gLii
Li  1 isos 2
Li-6    5
Li-7    99995

grp gBei
Be  1   isos    1
Be-9    1

grp gCi
C  1   isos    1
C-12    1

grp gFi
F   1   isos    1
F-19    1

grp gLiFo
Li  1
F   1

grp gBeF2o
Be  1
F   2

grp gFo
F   1

grp gThF4o
Th  1
F   4

grp gUF4o
U   1
F   4

grp gF
```

```
F   1

grp gAllF   sum
gF-li   1
gF-be   1
gF-th   1
gF-u    1
gF      1

% ------ End of Groups

% ------ Conditions section

conditions Cfuel
rng gLiS  min 0.26 max 0.30
rng gLi min 0.0 max 1.0
rng gAllLi  min 0.0 max 1.0
rto gF-li  val 1 grp2 gLiS
rng gBe min 0.061 max 0.065
rto gF-be val 2 grp2 gBe
rto gF-th val 4 grp2 gTh232
rto gF-u val 4 grp2 gU
rng gAllF   min 0.0 max 1.0
rng gF min 0.0 max 1.0
cnt cnt_table
oxi oxid_control min -0.0002  max -0.0001
opt dir min type action streams

% ------ End of Conditions Section

% ------ Control Section

control cnt_table
Li
Be
C
F
Th-232
U

% ------ End of Control Section

% ------ Begining of Feed/Removal/Redox/Reac section
```

```
stream to fuel group gLii      type    feed    form   cont
stream from fuel group gLiFo      type    remv    form   cont
stream to fuel group gBei       type    feed    form   cont
stream from fuel group gBeF2o   type    remv    form   cont
stream to fuel group gFi        type    feed    form   cont
stream from fuel group gFo        type    remv    form   cont
stream to fuel group gThF4i    type    reac    form   cont
stream from fuel group gThF4o    type    reac    form   cont
stream to fuel group gUF4i     type    reac    form   cont
stream from fuel group gUF4o    type    reac    form   cont
stream from fuel rem proc    type    remv    form    cont frac  2.19795529E-09
stream from fuel rem natural_rem    type    remv    form    prop  frac 0.023105


% ------ End of Streams

% ------ Table-class stream definitions

removal natural_rem
He  1
Ne  1
Ar  1
Kr  1
Nb  1
Mo  1
Tc  1
Ru  1
Rh  1
Pd  1
Ag  1
Sb  1
Te  1
Xe  1
Rn  1

removal proc
B  1
N   1
C   1
O   1
Na  1
Mg  1
Al  1
Si  1
```

```
P   1
S   1
Cl  1
K   1
Ca  1
Sc  1
Ti  1
V   1
Cr  1
Mn  1
Fe  1
Co  1
Ni  1
Cu  1
Lu  1
Hf  1
Ta  1
W   1
Re  1
Os  1
Ir  1
Pt  1
Au  1
Hg  1
Tl  1
Pb  1
Bi  1
Po  1
At  1
Fr  1
Ra  1
Lr  1
Rf  1
Db  1
Sg  1
Bh  1
Hs  1
Mt  1
Ac  1
Pa  1
Zn 1
Ga 1
Ge 1
As 1
```

```
Se 1
Br 1
Rb 1
Sr 1
Y  1
Zr 1
Cd 1
In 1
Sn 1
I  1
La 1
Ce 1
Pr 1
Nd 1
Pm 1
Sm 1
Eu 1
Gd 1
Tb 1
Dy 1
Ho 1
Er 1
Tm 1
Yb 1
Cs 1
Ba 1

% --- Oxidation Table

oxidation oxid_control
H   1
Li  1
Na  1
K   1
Rb  1
Cs  1
Fr  1
Be  2
Mg  2
Ca  2
Sr  2
Ba  2
Ra  2
Sc  3
```

```
Y   3
Ti  4
Zr  4
Hf  4
Rf  4
V   5
Nb  5
Ta  5
Db  5
Cr  3
Mo  4
W   4
Sg  6
Mn  2
Tc  4
Re  4
Bh  7
Mn  2
Tc  4
Re  4
Bh  7
Fe  2
Ru  3
Os  4
Hs  8
Co  2
Rh  3
Ir  3
Mt  7
Ni  2
Pd  2
Pt  2
Ds  6
Cu  2
Ag  1
Au  3
Rg  5
Zn  2
Cd  2
Hg  1
B   3
Al  3
Ga  3
In  3
```

```
Ti  1
C   4
Si  4
Ge  4
Sn  4
Pb  2
N   3
P   3
As  3
Sb  3
Bi  3
O   -2
S   -2
Se  -2
Te  -2
Po  -2
F   -1
Cl  -1
Br  -1
I   -1
At  -1
He  0
Ne  0
Ar  0
Kr  0
Xe  0
Rn  0
La  3
Ce  3
Pr  3
Nd  3
Pm  3
Sm  3
Eu  2
Gd  3
Tb  3
Dy  3
Ho  3
Er  3
Tm  3
Yb  3
Lu  3
Ac  3
Th  4
```

```
Pa  4
U   4
Nb  4
Pu  4
Am  4
Cm  4
Bk  4
Cf  4
Es  4
Fm  4
Md  4
No  4
Lr  3

% --- End Oxidation Table

% --- Boundary condition

set bc 3

% --- Neutron population and criticality cycles:

set pop 10000 40 60

% --- Fission Product Cut Off

set fpcut 1E-8

% --- Stable Nuclide Cut Off

set stabcut 1E-12

% --- Set Burnup Solution Method ( CRAM )

set bumode 2

% --- Set Predictor Corrector Usage

set pcc "LELI" [1 1]

% --- Set flux for burnup

set power 1E+8
```

```
% --- Set Inventory for Burnup Report

set inventory all

% --- Set xscalc mode

set xscalc 1

% --- Set materials printing mode

set printm 1

% --- Set Optimization Mode

set opti 4

% --- Bunup Intervals

dep daystep
1
1
2
3
5
8
13
21
34
...
34


% ------ ADER Reactivity Targets
kmin 1.0000000
kmax 1.0100000

% ------ ADER Iteration Set
set ader_trans_iter 1

% --- Plot
```